hroug '23 annual conference

{TRIOLOGY

Transfer bad PL/SQL into good

ROVINJ
18.10.2023

Maik Becker

maik.becker@triology.de

@beckerman_maik

Alexandra Welzel

alexandra.welzel@triology.de

@alexandrawelzel

Business Unit Manager @TRIOLOGY

> 20 years of experience in software development

> 10 years of experience with Oracle DB and APEX

Oracle ACE Associate

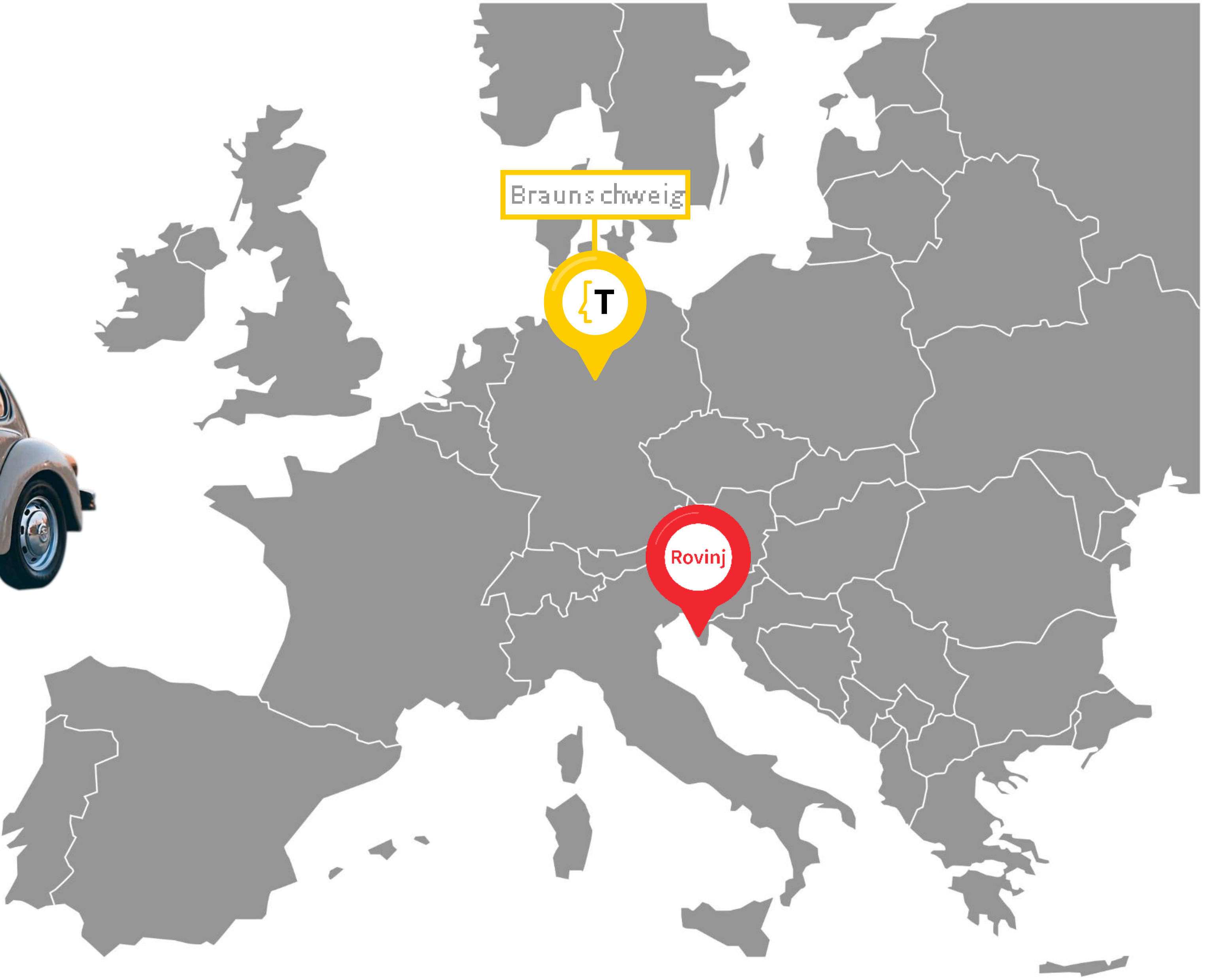
Data Consultant @TRIOLOGY

focused on Oracle DB and Oracle APEX

6 years of experience in database development

Love Oracle ā'pěks

Where are we from?



Safety instructions

Most examples are based on actual source code we found somewhere over the years

Names and authors have been erased, where we used unmodified examples

Some examples may be made up but are based on real program code, we found elsewhere

If you identify your own code, keep cool, stay calm

Some examples have been strongly modified

We do not want to embarrass anybody - It is all about learning!

Unused or unnecessary code

Unused or unnecessary code

#1 – various useless lines

```
PROCEDURE GetFBTDocuments (
    infinalbtid     IN      NUMBER,
    outresult       OUT  ref_cursor_type,
    outresultmsg OUT  VARCHAR2)
IS
    step  NUMBER;
BEGIN
    step := 1;
    OPEN outresult FOR
    SELECT (...) FROM (...) WHERE (...) = infinalbtid;
-- exception ------------------------------------------------------------
EXCEPTION
WHEN OTHERS THEN
    BEGIN
        ROLLBACK;
        outresultmsg := SUBSTR (SQLERRM, 1, 450) || '. SQLCode: ' || SQLCODE;
        DBMS_OUTPUT.put_line (outresultmsg);
    END;
END GetFBTDocuments;
```

better

```
PROCEDURE GetFBTDocuments (
    infinalbtid     IN      NUMBER,
    outresult       OUT  ref_cursor_type,
    outresultmsg OUT  VARCHAR2)
IS
BEGIN
    OPEN outresult FOR
    SELECT (...) FROM (...) WHERE (...) = infinalbtid;
EXCEPTION
WHEN OTHERS THEN
        ROLLBACK;
        outresultmsg := SUBSTR (SQLERRM, 1, 450) || '. SQLCode: ' || SQLCODE;
        -- consider logging here instead of console output!
        -- also SQLERRM will likely give incomplete error message
END GetFBTDocuments;
```

TRIOLOGY

Unused or unnecessary code

#2 – unused variables in declaration section

```
PROCEDURE UpdateStufe1 (inFinalBTID IN NUMBER, inEvn IN VARCHAR2, inType IN VARCHAR2,
    inText IN VARCHAR2, inUserId IN VARCHAR2, inUserName IN VARCHAR2, outresult OUT NUMBER, outresultmsg OUT VARCHAR2)
IS
    step        NUMBER;
    ~~count       NUMBER;~~
    ~~cnt2        NUMBER;~~
    ~~firstFBTID  NUMBER;~~
    ~~isOpen      BOOLEAN;~~
BEGIN
    outresult := 0;
    outresultmsg := '';

    step := 1;
    UPDATE dactrl.dco_bt_status s SET (...) WHERE s.id = inFinalBTID AND s.fbt_status != 1;

    IF (SQL%ROWCOUNT <> 1) THEN
        (…)
    END IF;

    step := 2;
    DCO_COM.SaveComment (inEvn, inType, inText, inUserId, inUserName, outresult, outresultmsg);

    IF (outresult < 0) THEN
        (…)
    END IF;

    outresult := step;
END UpdateStufe1;
```

Unused or unnecessary code

#3 – unreachable code

```
PROCEDURE ForwardFBTCheck
    (inDealerID          IN    VARCHAR2
    ,inEVN               IN    VARCHAR2
    ,inFBTID             IN    NUMBER
    ,outIsForwardable    OUT   NUMBER)
is
begin
    insert into dco_bt (FBTID, EVN, DealerID)
    values (inFBTID, inEVN, inDealerID);

    IF (SQL%ROWCOUNT != 1) THEN

        -- do some stuff here

        outIsForwardable := -1;
    END IF;

    -- do other stuff here

    outIsForwardable := 1;

end ForwardFBTCheck;
```

better

```
PROCEDURE ForwardFBTCheck
    (inDealerID          IN    VARCHAR2
    ,inEVN               IN    VARCHAR2
    ,inFBTID             IN    NUMBER
    ,outIsForwardable    OUT   NUMBER)
is
begin
    begin
        insert into dco_bt (FBTID, EVN, DealerID)
        values (inFBTID, inEVN, inDealerID);

        outIsForwardable := 1;

    exception
    when others then
        outIsForwardable := -1;
    end;

    -- do other stuff here

end ForwardFBTCheck;
```

TRIOLOGY

Unused or unnecessary code

#4 – unreachable code (exception)

```
procedure UpdateFinalBtStatusAfterFIT
   (infinalbtid      IN    NUMBER
   ,inForwardedBy    IN    VARCHAR2
   ,inForwardedTo    IN    VARCHAR2
   ,outresult        OUT   NUMBER
   ,outresultmsg     OUT   VARCHAR2)
is
begin
  outresultmsg := '';

  update dco_bt
   set ForwardedBy = inForwardedBy, ForwardedTo = inForwardedTo
   where finalbtid = infinalbtid;

  outresult := 0;
exception
when no_data_found then
  outresult := -1;
  outresultmsg := substr(SQLCODE || ': ' || SQLERRM, 1, 450);

end UpdateFinalBtStatusAfterFIT;
```

better

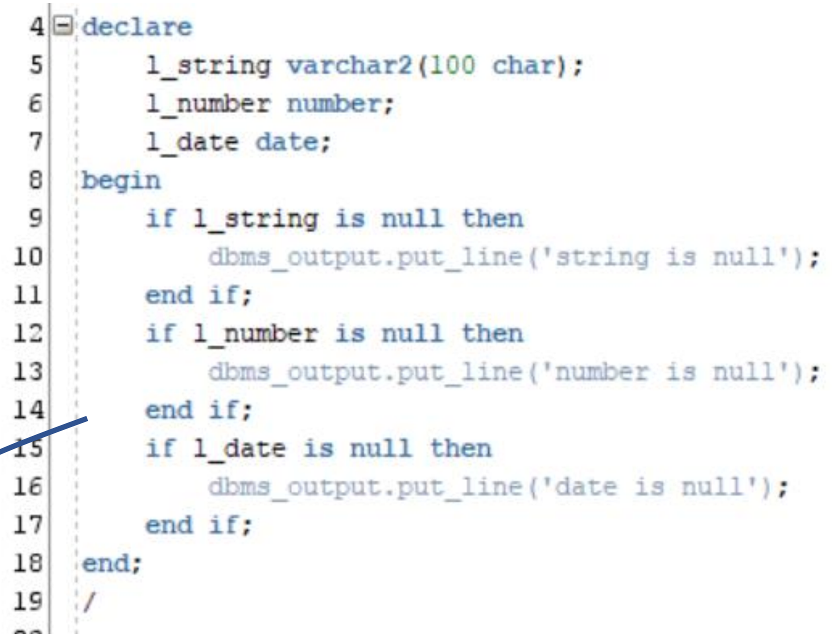
```
procedure UpdateFinalBtStatusAfterFIT
   (infinalbtid      IN    NUMBER
   ,inForwardedBy    IN    VARCHAR2
   ,inForwardedTo    IN    VARCHAR2
   ,outresult        OUT   NUMBER
   ,outresultmsg     OUT   VARCHAR2)
is
begin
  outresultmsg := '';

  update dco_bt
   set ForwardedBy = inForwardedBy, ForwardedTo = inForwardedTo
   where finalbtid = infinalbtid;

   if SQL%ROWCOUNT = 1 then
      outresult := 0;
   else
      outresult := -1;
      outresultmsg := 'something went wrong while updating a single row!';
   end if;

end UpdateFinalBtStatusAfterFIT;
```

TRIOLOGY

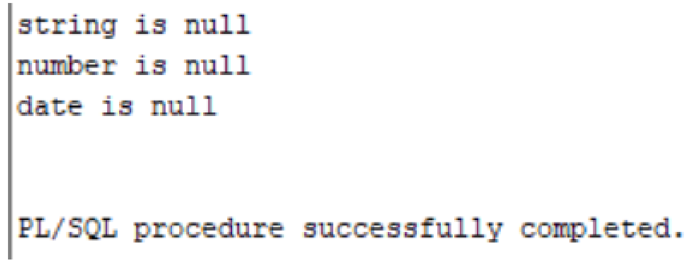
Unused or unnecessary code

#5 – initialization of variables with null

```
function convert_txt_to_html ( in_txt_message in varchar2 )
            return varchar2
is
            l_html_message  varchar2(32767) default in_txt_message;
            l_temp_url varchar2(32767) ~~:= null;~~
            l_length number ~~default null;~~
begin
            (...)
            return l_html_message;
end convert_txt_to_html;
```



```
declare
    l_string varchar2(100 char);
    l_number number;
    l_date date;
begin
    if l_string is null then
        dbms_output.put_line('string is null');
    end if;
    if l_number is null then
        dbms_output.put_line('number is null');
    end if;
    if l_date is null then
        dbms_output.put_line('date is null');
    end if;
end;
/
```



```
string is null
number is null
date is null


PL/SQL procedure successfully completed.
```

Unused or unnecessary code

#6 – count(*) before loop

```
procedure map_organisations(in_costcenter in number)
is
    l_count number;
begin
    select count(*)
    into l_count
    from organisations
    where coc_id = in_costcenter;

    if l_count > 0 then
        for rec in (select * from organisations where coc_id = in_costcenter)
        loop
            -- do_something here

        end loop;
    end if;
end map_organisations;
```

better

```
procedure map_organisations(in_costcenter in number)
is
begin
    for rec in (select * from organisations where coc_id = in_costcenter)
    loop
        -- do_something here

    end loop;
end map_organisations;
```

Unused or unnecessary code

#7 – select from dual for almost everything

```
FUNCTION next_store_seq
    RETURN NUMBER
IS
    next_id   NUMBER;
BEGIN
    SELECT   TRUNC (DBMS_RANDOM.VALUE (1000, 9999))
           * bit_shift + cbo_datastore_seq.NEXTVAL
      INTO next_id
      FROM DUAL;

    RETURN next_id;
END next_store_seq;
```

better

```
FUNCTION next_store_seq
    RETURN NUMBER
IS
BEGIN
    RETURN TRUNC (DBMS_RANDOM.VALUE (1000, 9999))
        * bit_shift + cbo_datastore_seq.NEXTVAL;
END next_store_seq;
```

TRIOLOGY

Unused or unnecessary code

#7 – select from dual for almost everything

```
declare
  l_date date;
  l_timestamp timestamp;
  l_number number;
  l_string varchar2(100 char);
begin
  select sysdate into l_date from dual;
  select add_months(sysdate, 12) into l_date from dual;
  select systimestamp into l_timestamp from dual;

  select seq.nextval into l_number from dual;
  select round(1/3, 2) into l_number from dual;
  select DBMS_RANDOM.VALUE (1000, 9999) into l_number from dual;
  select length('my string') into l_number from dual;

  select substr('my string', 4, 6) into l_string from dual;
  select rtrim('my string!!!!!', '!') into l_string from dual;
  select replace('my string!', '!', '?') into l_string from dual;

  -- and so on ...

end;
```

better

```
declare
  l_date date;
  l_timestamp timestamp;
  l_number number;
  l_string varchar2(100 char);
begin
  l_date := sysdate;
  l_date := add_months(sysdate, 12);
  l_timestamp :=  systimestamp;

  l_number :=  seq.nextval;
  l_number := round(1/3, 2);
  l_number := DBMS_RANDOM.VALUE (1000, 9999);
  l_number := length('my string');

  l_string := substr('my string', 4, 6);
  l_string := rtrim('my string!!!!!', '!');
  l_string := replace('my string!', '!', '?’);

  -- and so on ...

end;
```

Unused or unnecessary code

carefully think about, what you really need and what you don’t need

carefully think about, what can really happen and what will not be possible

eleminate every…
…variable you do not use
…piece of code that has no effect
…piece of code that will never execute

avoid unnecessary *select … from dual*

/* */

do not comment, what can be seen obviously

do not write errors to the console output

keep your code clean!

refactor your code!

Exception Handling

Exception Handling

#1 – ignore every exception

```
FUNCTION get_valid_list(in_list_key VARCHAR2
    ,in_valid_date DATE DEFAULT NULL
    ,in_client VARCHAR2 DEFAULT 'Mandant_DE'
    ,in_country VARCHAR2 DEFAULT 'Germany')
RETURN NUMBER
IS
    valid_list isa_paramlist.id%TYPE;
    valid_date DATE;
    client_id  NUMBER;
BEGIN
    IF in_valid_date IS NOT NULL THEN
        valid_date := in_valid_date;
    ELSE
        valid_date := SYSDATE;
    END IF;

    SELECT A.id INTO client_id FROM (...) WHERE (...);
    SELECT id INTO valid_list FROM (...) WHERE (...);

    RETURN valid_list;
exception
    WHEN others THEN -- caution handles all exceptions
    RETURN NULL;
END get_valid_list;
```

better

```
FUNCTION get_valid_list(in_list_key VARCHAR2
    ,in_valid_date DATE DEFAULT NULL
    ,in_client VARCHAR2 DEFAULT 'Mandant_DE'
    ,in_country VARCHAR2 DEFAULT 'Germany')
RETURN NUMBER
IS
    valid_list isa_paramlist.id%TYPE;
    valid_date DATE;
    client_id  NUMBER;
BEGIN
    IF in_valid_date IS NOT NULL THEN
        valid_date := in_valid_date;
    ELSE
        valid_date := SYSDATE;
    END IF;

    SELECT A.id INTO client_id FROM (...) WHERE (...);
    SELECT id INTO valid_list FROM (...) WHERE (...);

    RETURN valid_list;
exception
    WHEN others THEN
    -- at least logging!
    logger.log_error('unknown error');
    RETURN NULL;

END get_valid_list;
```

TRIOLOGY

Exception Handling

#1 – ignore every exception

```
FUNCTION get_valid_list(in_list_key VARCHAR2
    ,in_valid_date DATE DEFAULT NULL
    ,in_client VARCHAR2 DEFAULT 'Mandant_DE'
    ,in_country VARCHAR2 DEFAULT 'Germany')
RETURN NUMBER
IS
    valid_list isa_paramlist.id%TYPE;
    valid_date DATE;
    client_id  NUMBER;
BEGIN
    IF in_valid_date IS NOT NULL THEN
        valid_date := in_valid_date;
    ELSE
        valid_date := SYSDATE;
    END IF;

    SELECT A.id INTO client_id FROM (...) WHERE (...);
    SELECT id INTO valid_list FROM (...) WHERE (...);

    RETURN valid_list;
exception
    WHEN others THEN -- caution handles all exceptions
    RETURN NULL;
END get_valid_list;
```

much better

```
FUNCTION get_valid_list(in_list_key VARCHAR2
    ,in_valid_date DATE DEFAULT NULL
    ,in_client VARCHAR2 DEFAULT 'Mandant_DE'
    ,in_country VARCHAR2 DEFAULT 'Germany')
RETURN NUMBER
IS
    valid_list isa_paramlist.id%TYPE;
    valid_date DATE;
    client_id  NUMBER;
BEGIN
    IF in_valid_date IS NOT NULL THEN
        valid_date := in_valid_date;
    ELSE
        valid_date := SYSDATE;
    END IF;

    SELECT A.id INTO client_id FROM (...) WHERE (...);
    SELECT id INTO valid_list FROM (...) WHERE (...);

    RETURN valid_list;
exception
    WHEN no_data_found THEN
    -- nothing was found
    RETURN NULL;

    WHEN others THEN
    logger.log_error('unknown error');
    RETURN NULL;

END get_valid_list;
```

TRIOLOGY

Exception Handling

#1 – ignore every exception

```
FUNCTION get_valid_list(in_list_key VARCHAR2
    ,in_valid_date DATE DEFAULT NULL
    ,in_client VARCHAR2 DEFAULT 'Mandant_DE'
    ,in_country VARCHAR2 DEFAULT 'Germany')
RETURN NUMBER
IS
    valid_list isa_paramlist.id%TYPE;
    valid_date DATE;
    client_id  NUMBER;
BEGIN
    IF in_valid_date IS NOT NULL THEN
        valid_date := in_valid_date;
    ELSE
        valid_date := SYSDATE;
    END IF;

    SELECT A.id INTO client_id FROM (...) WHERE (...);
    SELECT id INTO valid_list FROM (...) WHERE (...);

    RETURN valid_list;
exception
    WHEN others THEN -- caution handles all exceptions
    RETURN NULL;
END get_valid_list;
```

also ok (sometimes)

```
FUNCTION get_valid_list(in_list_key VARCHAR2
    ,in_valid_date DATE DEFAULT NULL
    ,in_client VARCHAR2 DEFAULT 'Mandant_DE'
    ,in_country VARCHAR2 DEFAULT 'Germany')
RETURN NUMBER
IS
    valid_list isa_paramlist.id%TYPE;
    valid_date DATE;
    client_id  NUMBER;
BEGIN
    IF in_valid_date IS NOT NULL THEN
        valid_date := in_valid_date;
    ELSE
        valid_date := SYSDATE;
    END IF;

    SELECT A.id INTO client_id FROM (...) WHERE (...);
    SELECT id INTO valid_list FROM (...) WHERE (...);

    RETURN valid_list;
exception
    WHEN others THEN
    -- it is ok, to ignore all errors here, because ...
    RETURN NULL;

END get_valid_list;
```

Exception Handling

#2 – ignore every exception

Just writing something to the output buffer
is nothing but ignoring the exception!
Who will ever read it?

```
PROCEDURE DataProtectionAction( ... )
AS
    (...)
BEGIN
    IF in_advisorid IS NOT NULL THEN
        BEGIN
            SELECT (...) INTO (...) FROM (...) WHERE (...) ;
            dbms_output.put_line('l_advisorID ='||l_advisorID);
        EXCEPTION
        WHEN NO_DATA_FOUND THEN
            BEGIN
                SELECT (...) INTO (...) FROM (...) WHERE (...) ;
            EXCEPTION
            WHEN OTHERS THEN
                dbms_output.put_line('exeption l_advisorID ='||l_advisorID);
            END ;
        WHEN OTHERS THEN
            NULL;
        END ;
    END IF ;

    IF in_object_type = 2 THEN
        (...)
        WHILE in_obj_ids.existsNode('//fkey[' || l_count || ']') = 1 LOOP
            (...)
            BEGIN
                (...)
            EXCEPTION
            WHEN OTHERS THEN
                dbms_output.put_line( 'Exception bei ermitteln der person id '|| l_fkey || ' '|| sqlerrm );
            END ;
        END LOOP ;
    ELSE
        dbms_output.put_line(  'only for persons' );
    END IF ;
END DataProtectionAction ;
```

TRIOLOGY

Exception Handling

#3 – just re-raise

Worse than doing nothing!
Original line number of exception will get lost.
No value at all.

```
PROCEDURE my_proc(in_id number)
IS
BEGIN
    -- do stuff
EXCEPTION
    WHEN OTHERS THEN
        RAISE;
END my_proc;
```

better

```
PROCEDURE my_proc(in_id number)
IS
BEGIN
    -- do stuff
EXCEPTION
    WHEN OTHERS THEN
        logger.log_error ('unknown error ... ');
        RAISE;
END my_proc;
```

TRIOLOGY

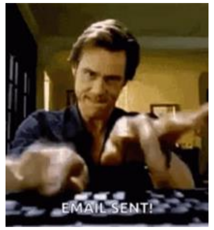
Exception Handling

#3 – block application because of logging error

```
procedure log_message(in_text in varchar2)
is
    pragma autonomous_transaction;
begin
    insert
    into its_process_log(
        id, created_date, message, parallel_deg, cln_id)
    values(
        to_number(to_char(systimestamp,'YYYYMMDDHH24MISSFF6'))
        , sysdate
        , '[' || trim(to_char(g_parallel)) || '] ' || in_text
        , g_parallel
        , g_cln_id);
    commit;
exception when others
then
    raise_application_error (-20999, 'Cannot write log message. ' || sqlerrm);
end log_message;
```

Caller most likely will not expect a logging procedure returning with an exception

Do something else … But do not block the entire application, just because your logging does not work!



send an email to the admin

turn on warning lights

let it rain

use a monitoring to check your logging

Exception Handling

#4 – SQLERRM and SQLCODE

```
FUNCTION GetFBTDocumentName(infinalbtid    IN NUMBER)
    RETURN VARCHAR2
IS
    l_doc_name varchar2(100 char);
    l_message varchar2(4000 char);
BEGIN

    SELECT name into l_doc_name
    FROM dco_documents
    WHERE id = infinalbtid;
    return l_doc_name;

EXCEPTION
WHEN OTHERS THEN

    l_message := SUBSTR (SQLERRM, 1, 450) || '. SQLCode: ' || SQLCODE;
    DBMS_OUTPUT.put_line (l_message);

   return null;
END GetFBTDocumentName;
```

```
ORA-01403: no data found. SQLCode: 100
```

better

```
FUNCTION GetFBTDocumentName(infinalbtid    IN NUMBER)
    RETURN VARCHAR2
IS
    l_doc_name varchar2(100 char);
    l_message varchar2(4000 char);
BEGIN

    SELECT name into l_doc_name
    FROM dco_documents
    WHERE id = infinalbtid;
    return l_doc_name;

EXCEPTION
WHEN OTHERS THEN
    l_message :=
        dbms_utility.FORMAT_ERROR_STACK
        || dbms_utility.FORMAT_ERROR_BACKTRACE;

    logger.log_error (l_message);
    DBMS_OUTPUT.put_line (l_message);

   return null;
END GetFBTDocumentName;
```

```
ORA-01403: no data found
ORA-06512: at "CRMOWN.GETFBTDOCUMENTNAME", line 7
```

TRIOLOGY

Exception Handling

never ignore all exceptions with *when others then null*

try to use named exceptions and handle most likely exceptions separately from unknown errors

for example *no_data_found*, which often is more a data condition than an error

/* . . . */

if really using *when others then null*:
write a comment, why everything must be ignored here!

do not just write exceptions to the console, use logging instead

never re-raise without logging!

never block your application because of erroneous logging!

TRIOLOGY

Code Control Structure

Code Control Structure

#1 – use of GOTO

```
PROCEDURE SaveDigitalOrPaperBT(inevn varchar2, inisdigital number, outResult out number)
IS
    step number;
BEGIN
    step := 1;
    IF (inevn IS NULL OR inisdigital < 0 OR inisdigital > 1) THEN
        step := -step;
        GOTO ende;
    END IF;

    step := 2;
    (...)

    step := 3;
    UPDATE dco_bt SET isDigital = inisdigital WHERE evn = inevn;
    IF (SQL%ROWCOUNT = 0) THEN
        step := -step;
        GOTO step6;
    END IF;

    -- step 4, 5

    <<step6>>
    IF (inisdigital = 1) THEN
        step := 6;
     (...)
    END IF;

  -- step 7, 8, 9

    <<ende>>
    outresult := step;
END SaveDigitalOrPaperBT;
```

better

```
PROCEDURE SaveDigitalOrPaperBT(inevn varchar2, inisdigital number, outResult out number)
IS
    step number;
BEGIN
    step := 1;
    IF (inevn IS NULL OR inisdigital < 0 OR inisdigital > 1) THEN
        step := -step;

    ELSE
        step := 2;
        (...)

        step := 3;
        UPDATE dco_bt SET isDigital = inisdigital WHERE evn = inevn;
        IF (SQL%ROWCOUNT = 0) THEN
            step := -step;

        ELSE
            -- step 4, 5

        END IF;

        IF (inisdigital = 1) THEN
            step := 6;
            (...)
        END IF;

        -- step 7, 8, 9
    END IF;

    outresult := step;
END SaveDigitalOrPaperBT;
```

TRIOLOGY

Code Control Structure

#2 – break a loop with return

```
function has_final_status(in_group in ecm_activities.group_no%type)
  return boolean
is
begin
  for rec in (select * from ecm_activities where group_no = in_group)
  loop
    if rec.status in (3, 7, 9)
    then
      return true;
    end if;
  end loop;

  return false;

end has_final_status;
```

good

```
function has_final_status(in_group in ecm_activities.group_no%type)
  return boolean
is
  l_return boolean := false;
begin
  for rec in (select * from ecm_activities where group_no = in_group)
  loop
    if rec.status in (3, 7, 9)
    then
      l_return := true;
    end if;
  end loop;

  return l_return;

end has_final_status;
```

TRIOLOGY

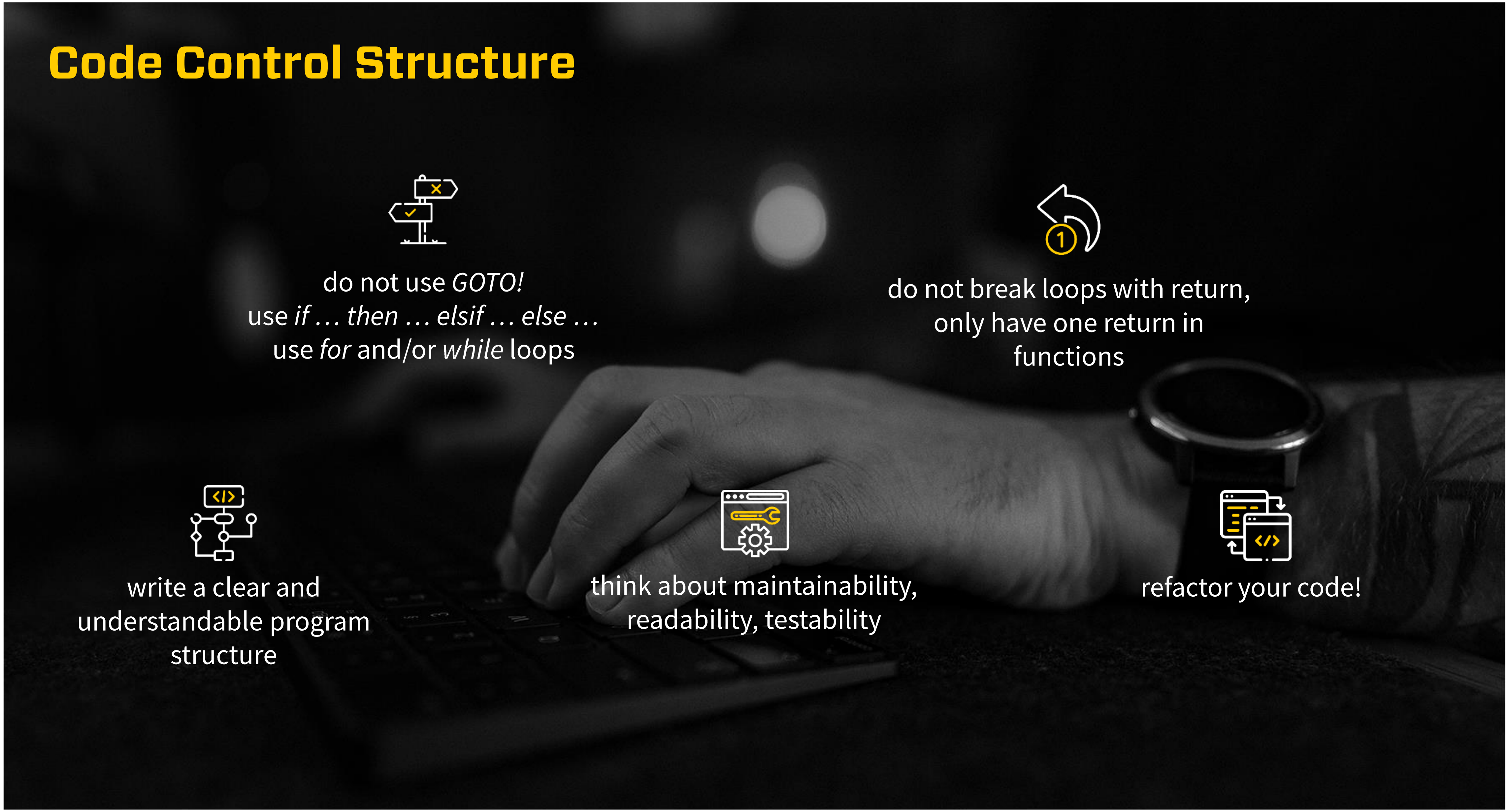
Code Control Structure

#2 – break a loop with return

```
function has_final_status(in_group in ecm_activities.group_no%type)
  return boolean
is
begin
  for rec in (select * from ecm_activities where group_no = in_group)
  loop
    if rec.status in (3, 7, 9)
    then
      return true;
    end if;
  end loop;

  return false;

end has_final_status;
```

better

```
function has_final_status(in_group in ecm_activities.group_no%type)
  return boolean
is
  l_return boolean := false;
  l_count number;
begin
  select count(*) into l_count
  from ecm_activities
  where group_no = in_group and status in (3, 7, 9);

  if l_count > 0 then l_return := true;
  else l_return := false;
  end if;

  return l_return;

end has_final_status;
```



Code Control Structure

do not use *GOTO!*
use *if … then … elsif … else …*
use *for* and/or *while* loops

do not break loops with return,
only have one return in
functions

write a clear and
understandable program
structure

think about maintainability,
readability, testability

refactor your code!

General Programming Issues

TRIOLOGY

General Programming Issues

#1 – no prefix for input parameters

```
procedure set_stock(
  part_name     in ugl_parts.part_name%type,
  stock         in ugl_parts.stock%type)
is
begin
  update ugl_parts
  set stock = stock
  where part_name = part_name;
end set_stock;
```

General Programming Issues

#1 – no prefix for input parameters

```
procedure set_stock(
   part_name      in ugl_parts.part_name%type,
   stock          in ugl_parts.stock%type)
is
begin
   update ugl_parts
   set  :new  =  :old
   where              1 = 1             ;
end set_stock;
```

better

```
procedure set_stock(
   in_part_name   in ugl_parts.part_name%type,
   in_stock       in ugl_parts.stock%type)
is
begin
   update ugl_parts
   set stock = in_stock
   where part_name = in_part_name;
end set_stock;
```

TRIOLOGY

General Programming Issues

#1 – no prefix for input parameters

```
procedure set_stock(
   part_name     in ugl_parts.part_name%type,
   stock         in ugl_parts.stock%type)
is
begin
   update ugl_parts
   set  stock = set_stock.stock
   where part_name = set_stock.part_name;
end set_stock;
```

preferred

```
procedure set_stock(
   in_part_name   in ugl_parts.part_name%type,
   in_stock       in ugl_parts.stock%type)
is
begin
   update ugl_parts
   set stock = in_stock
   where part_name = in_part_name;
end set_stock;
```

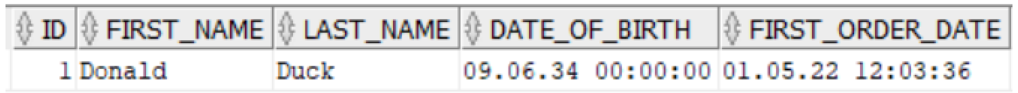
General Programming Issues

#2 – call by position vs. call by name

```
procedure set_customer(
    in_id in number
   ,in_first_name in varchar2
   ,in_last_name in varchar2
   ,in_first_order in date
   ,in_date_of_birth in date)
is
begin
   insert into sro_customer (id, first_name, last_name, first_order_date, date_of_birth)
   values (in_id, in_first_name, in_last_name, in_first_order, in_date_of_birth);
end set_customer;
```

Call by position

```
begin
   set_customer(1, 'Donald', 'Duck', sysdate, to_date('1934-06-09', 'yyyy-mm-dd'));
end;
```



ID	FIRST_NAME	LAST_NAME	DATE_OF_BIRTH	FIRST_ORDER_DATE
1	Donald	Duck	09.06.34 00:00:00	01.05.22 12:03:36

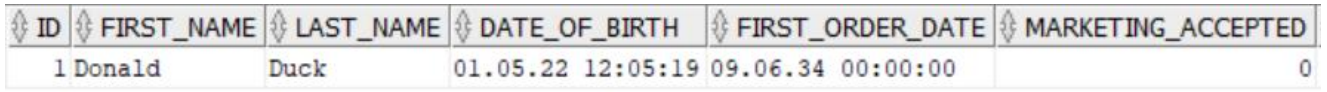
General Programming Issues

#2 – call by position vs. call by name

```
procedure set_customer(
    in_id in number
   ,in_first_name in varchar2
   ,in_last_name in varchar2
   ,in_date_of_birth in date
   ,in_first_order in date
   ,in_marketing_accepted in number default null)
is
begin
   insert into sro_customer (id, first_name, last_name, first_order_date,
                             date_of_birth, marketing_accepted)
   values (in_id, in_first_name, in_last_name, in_first_order,
                             in_date_of_birth, in_marketing_accepted);
end set_customer;
```

Call by position

```
begin
   set_customer(1, 'Donald', 'Duck', sysdate, to_date('1934-06-09', 'yyyy-mm-dd'));
end;
```



ID	FIRST_NAME	LAST_NAME	DATE_OF_BIRTH	FIRST_ORDER_DATE	MARKETING_ACCEPTED
1	Donald	Duck	01.05.22 12:05:19	09.06.34 00:00:00	0

better: call by name

```
begin
   set_customer(
      in_id => 1,
      in_first_name => 'Donald',
      in_last_name => 'Duck',
      in_first_order => sysdate,
      in_date_of_birth => to_date('1934-06-09', 'yyyy-mm-dd'));
end;
```

General Programming Issues

#3 – transaction isolation

```
declare
    l_apr_id             dca_app_responses.id%type;
    l_apo_id             dca_app_objects.id%type;
    l_app_save_objects   dca_applications.save_objects%type;
begin
    -- process some stuff here

    save_app_response(
        out_id               => l_apr_id,
        in_run_id            => in_run_id,
        in_app_id            => in_app_id,
        in_req_id            => in_req_id,
        in_request_action    => in_request_action,
        in_response_status   => in_response_status,
        in_response_message  => in_response_message,
        in_response_date     => in_response_date);

    -- do some kind of long and complex process ...

    -- and then ...

    if l_app_save_objects = 1 then
        save_app_objects(
            out_id      => l_apo_id,
            in_run_id   => in_run_id,
            in_app_id   => in_app_id,
            in_obj_id   => in_obj_id,
            in_apr_id   => l_apr_id);
    end if;
end;
```

```
procedure save_app_response(
  out_id out dca_app_responses.id%type,
  in_run_id in dca_app_responses.run_id%type,
  in_app_id in dca_app_responses.app_id%type,
  in_req_id in dca_app_responses.req_id%type,
  in_request_action in dca_app_responses.request_action%type,
  in_response_status in dca_app_responses.response_status%type,
  in_response_message in dca_app_responses.response_message%type,
  in_response_date in dca_app_responses.response_date%type)
is
begin

  insert into dca_app_responses ( … )
  values ( … )
  returning id into out_id;

end save_app_response;
```

TRIOLOGY

General Programming Issues

#3 – transaction isolation

```
declare
   l_apr_id            dca_app_responses.id%type;
   l_apo_id            dca_app_objects.id%type;
   l_app_save_objects  dca_applications.save_objects%type;
begin
   -- process some stuff here

   save_app_response(
      out_id              => l_apr_id,
      in_run_id           => in_run_id,
      in_app_id           => in_app_id,
      in_req_id           => in_req_id,
      in_request_action   => in_request_action,
      in_response_status  => in_response_status,
      in_response_message => in_response_message,
      in_response_date    => in_response_date);

   -- do some kind of long and complex process ...

   -- and then ...

   if l_app_save_objects = 1 then
      save_app_objects(
         out_id     => l_apo_id,
         in_run_id  => in_run_id,
         in_app_id  => in_app_id,
         in_obj_id  => in_obj_id,
         in_apr_id  => l_apr_id);
   end if;
end;
```

```
procedure save_app_objects(
  out_id out dca_app_objects.id%type,
  in_run_id in dca_app_objects.run_id%type,
  in_app_id in dca_app_objects.app_id%type,
  in_obj_id in dca_app_objects.obj_id%type,
  in_apr_id in dca_app_responses.id%type)
is
  PRAGMA AUTONOMOUS_TRANSACTION;
  l_last_request_action dca_app_objects.last_request_action%type;
  l_last_response_status dca_app_objects.last_response_status%type;
begin
  select request_action, response_status
  into l_last_request_action, l_last_response_status
  from dca_app_responses
  where id = in_apr_id;

  insert into dca_app_objects ( … )
  values ( … )
  returning id into out_id;

  commit;

end save_app_objects;
```

TRIOLOGY

General Programming Issues

#3 – transaction isolation

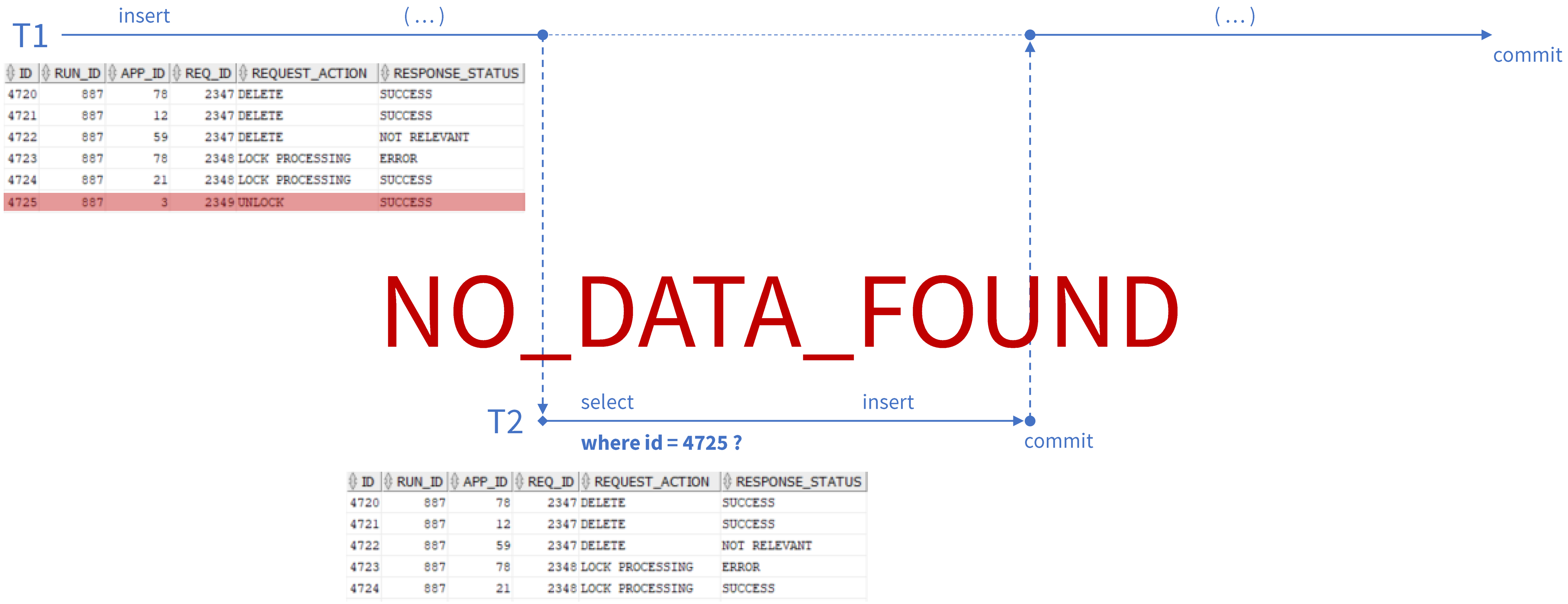
```
declare
   l_apr_id             dca_app_responses.id%type;
   l_apo_id             dca_app_objects.id%type;
   l_app_save_objects   dca_applications.save_objects%type;
begin
   -- process some stuff here

   save_app_response(
      out_id               => l_apr_id,
      in_run_id            => in_run_id,
      in_app_id            => in_app_id,
      in_req_id            => in_req_id,
      in_request_action    => in_request_action,
      in_response_status   => in_response_status,
      in_response_message  => in_response_message,
      in_response_date     => in_response_date);

   -- do some kind of long and complex process ...

   -- and then ...

   if l_app_save_objects = 1 then
      save_app_objects(
         out_id      => l_apo_id,
         in_run_id   => in_run_id,
         in_app_id   => in_app_id,
         in_obj_id   => in_obj_id,
         in_apr_id   => l_apr_id);
   end if;
end;
```

```
procedure save_app_objects(
   out_id out dca_app_objects.id%type,
   in_run_id in dca_app_objects.run_id%type,
   in_app_id in dca_app_objects.app_id%type,
   in_obj_id in dca_app_objects.obj_id%type,
   in_apr_id in dca_app_responses.id%type)
is
   PRAGMA AUTONOMOUS_TRANSACTION;
   l_last_request_action dca_app_objects.last_request_action%type;
   l_last_response_status dca_app_objects.last_response_status%type;
begin
   select request_action, response_status
   into l_last_request_action, l_last_response_status
   from dca_app_responses
   where id = in_apr_id;

   insert into dca_app_objects ( … )
   values ( … )
   returning id into out_id;

   commit;

end save_app_objects;
```

TRIOLOGY

General Programming Issues

#3 – transaction isolation



ID	RUN_ID	APP_ID	REQ_ID	REQUEST_ACTION	RESPONSE_STATUS
4720	887	78	2347	DELETE	SUCCESS
4721	887	12	2347	DELETE	SUCCESS
4722	887	59	2347	DELETE	NOT RELEVANT
4723	887	78	2348	LOCK PROCESSING	ERROR
4724	887	21	2348	LOCK PROCESSING	SUCCESS
4725	887	3	2349	UNLOCK	SUCCESS

ID	RUN_ID	APP_ID	REQ_ID	REQUEST_ACTION	RESPONSE_STATUS
4720	887	78	2347	DELETE	SUCCESS
4721	887	12	2347	DELETE	SUCCESS
4722	887	59	2347	DELETE	NOT RELEVANT
4723	887	78	2348	LOCK PROCESSING	ERROR
4724	887	21	2348	LOCK PROCESSING	SUCCESS

General Programming Issues

#3 – transaction isolation

Avoid using
PRAGMA AUTONOMOUS_TRANSACTION

There is (almost) no good reason
to use isolated transactions
Except: Logging!

Most other cases one can think about
are probably due to bad design

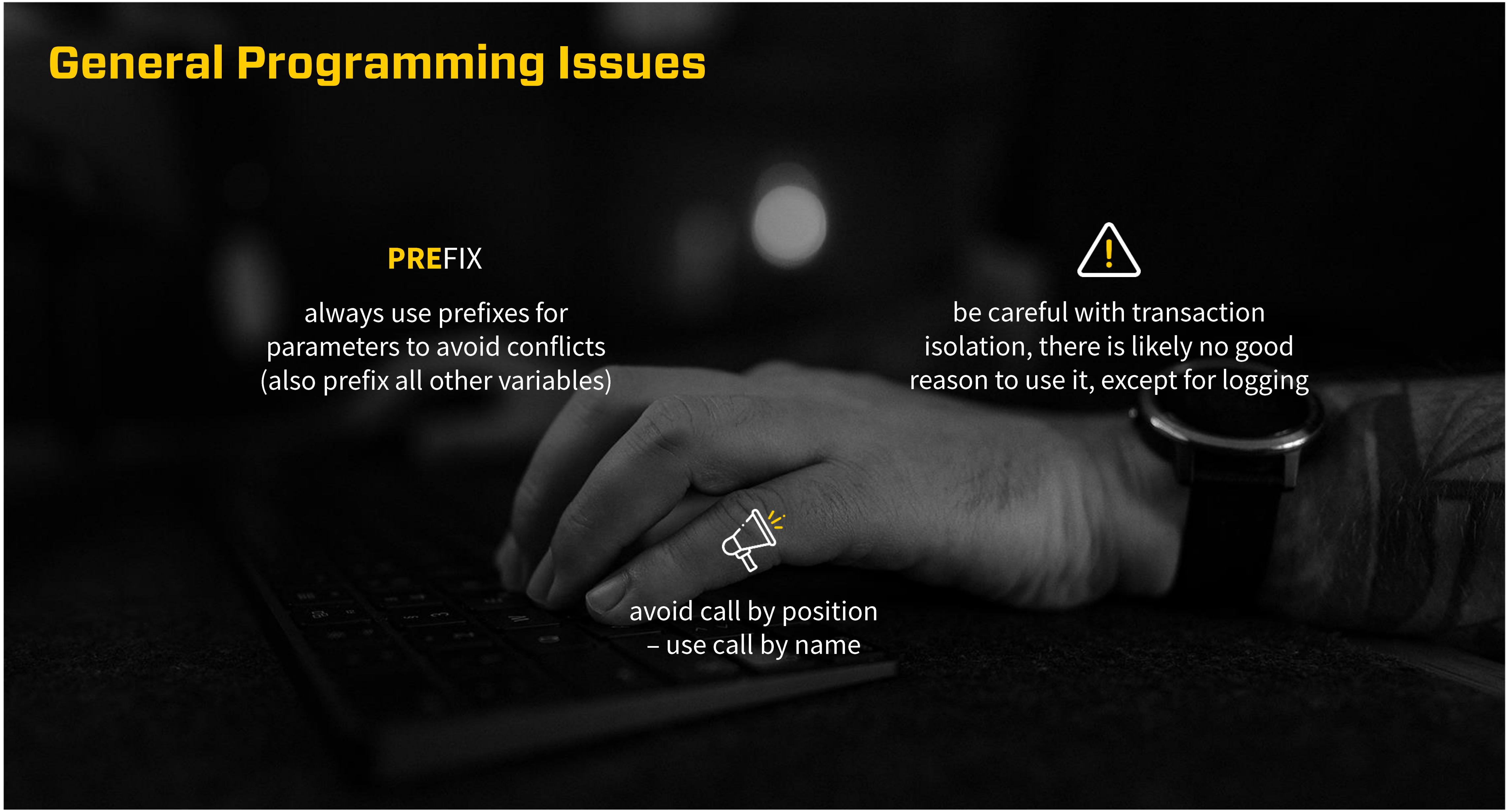
If there is a reason to use it anyway, be aware of:

⚠ Child transaction cannot see, what parent has done

⚠ Child transaction will not see data, manipulated by parent

⚠ Accessing resources in isolated transactions, which are held by the parent, will result into deadlock

⚠ In child transactions never access objects or data, which has been manipulated by the parent



General Programming Issues

PREFIX

always use prefixes for parameters to avoid conflicts (also prefix all other variables)

be careful with transaction isolation, there is likely no good reason to use it, except for logging

avoid call by position – use call by name

Potential Performance Issues

Potential Performance Issues

#1 – commit every row in a loop

```
procedure schedule_event(
  in_event_id in eco_event.id%type,
  in_schedule_time in eco_event.schedule_time%type)
is
begin
  update eco_event
  set schedule_time = in_schedule_time, scheduled = 1
  where id = in_event_id;

  for rec in (select * from eco_attendee where evt_id = in_event_id)
  loop
    if rec.applied = 1 then
      update eco_attendee
      set confirmed = 1
      where id = rec.id;

      commit;

    end if;
  end loop;
end schedule_event;
```

good

```
procedure schedule_event(
  in_event_id in eco_event.id%type,
  in_schedule_time in eco_event.schedule_time%type)
is
begin
  update eco_event
  set schedule_time = in_schedule_time, scheduled = 1
  where id = in_event_id;

  for rec in (select * from eco_attendee where evt_id = in_event_id)
  loop
    if rec.applied = 1
    then
      update eco_attendee
      set confirmed = 1
      where id = rec.id;
    end if;
  end loop;

  commit;

end schedule_event;
```

TRIOLOGY

Potential Performance Issues

#1 – commit every row in a loop

```
procedure schedule_event(
  in_event_id in eco_event.id%type,
  in_schedule_time in eco_event.schedule_time%type)
is
begin
  update eco_event
  set schedule_time = in_schedule_time, scheduled = 1
  where id = in_event_id;

  for rec in (select * from eco_attendee where evt_id = in_event_id)
  loop
   if rec.applied = 1 then
      update eco_attendee
      set confirmed = 1
      where id = rec.id;

      commit;

    end if;
  end loop;
end schedule_event;
```

better

```
procedure schedule_event(
  in_event_id in eco_event.id%type,
  in_schedule_time in eco_event.schedule_time%type)
is
begin
  update eco_event
  set schedule_time = in_schedule_time, scheduled = 1
  where id = in_event_id;

  update eco_attendee
  set confirmed = 1
  where evt_id = in_event_id
  and applied = 1;

  commit;

end schedule_event;
```

TRIOLOGY

Potential Performance Issues

#2 – execute a single DML inside a loop

```
declare
  cursor c1 is
    select id,addr_1,addr_2,city,state_code_id,zip,
           zip_plus4,country_id,updated_date,updated_by
     from (...)
    where (...);

  tmp_id              number;
  tmp_addr1           varchar2(35);
  tmp_addr2           varchar2(35);
  tmp_city            varchar2(25);
  tmp_state_id        number;
  tmp_zip             varchar2(5);
  tmp_zip4            varchar2(4);
  tmp_country_id      number;
  tmp_updated_date    date;
  tmp_updated_by      varchar2(10);
begin
  open c1;
  loop
    fetch c1 into tmp_id,tmp_addr1,tmp_addr2,tmp_city,tmp_state_id,
          tmp_zip,tmp_zip4,tmp_country_id,tmp_updated_date,tmp_updated_by;
    if c1%NOTFOUND then
      exit;
    end if;
    insert into LICENSEE_ADDRESS values (
          null,tmp_id,'3',tmp_addr1,tmp_addr2,tmp_city,tmp_state_id,tmp_country_id,
          tmp_zip,tmp_zip4,tmp_updated_date,tmp_updated_by,sysdate,'MIGRATION'
    );
  end loop;
  commit;
end;
```

better

```
declare
  cursor c_addresses is
    select id,addr_1,addr_2,city,state_code_id,
           zip,zip_plus4,country_id,updated_date,updated_by
     from (...)
    where (...);

  type addresses_t is table of c_addresses%rowtype;
  l_addr addresses_t;
begin
  open c_addresses;
  fetch c_addresses bulk collect into l_addr;
  close c_addresses;

  forall indx in 1 .. l_addr.count
    insert into LICENSEE_ADDRESS (
          id,
          <whatever this one is>,
          addr1,
          addr2,
          ... )
     values (
          l_addr(indx).id,
          '3‘,
          l_addr(indx).addr1,
          l_addr(indx).addr2,
          ... );
  commit;
end;
```

?

Thanks to Steven Feuerstein for providing this example

Potential Performance Issues

#2 – execute a single DML inside a loop

```
declare
  cursor c1 is
    select id,addr_1,addr_2,city,state_code_id,zip,
           zip_plus4,country_id,updated_date,updated_by
     from (...)
    where (...);

  tmp_id                number;
  tmp_addr1             varchar2(35);
  tmp_addr2             varchar2(35);
  tmp_city              varchar2(25);
  tmp_state_id          number;
  tmp_zip               varchar2(5);
  tmp_zip4              varchar2(4);
  tmp_country_id        number;
  tmp_updated_date      date;
  tmp_updated_by        varchar2(10);
begin
  open c1;
  loop
    fetch c1 into tmp_id,tmp_addr1,tmp_addr2,tmp_city,tmp_state_id,
          tmp_zip,tmp_zip4,tmp_country_id,tmp_updated_date,tmp_updated_by;
    if c1%NOTFOUND then
      exit;
    end if;
    insert into LICENSEE_ADDRESS values (
          null,tmp_id,'3',tmp_addr1,tmp_addr2,tmp_city,tmp_state_id,tmp_country_id,
          tmp_zip,tmp_zip4,tmp_updated_date,tmp_updated_by,sysdate,'MIGRATION'
    );
  end loop;
  commit;
end;
```

forall is not a loop!

better

```
declare
  cursor c_addresses is
    select id,addr_1,addr_2,city,state_code_id,
           zip,zip_plus4,country_id,updated_date,updated_by
     from (...)
    where (...);

  type addresses_t is table of c_addresses%rowtype;
  l_addr addresses_t;
begin
  open c_addresses;
  fetch c_addresses bulk collect into l_addr;
  close c_addresses;

  forall indx in 1 .. l_addr.count
    insert into LICENSEE_ADDRESS (
          id,
          <whatever this one is>,
          addr1,
          addr2,
          ... )
     values (
          l_addr(indx).id,
          '3‘,
          l_addr(indx).addr1,
          l_addr(indx).addr2,
          ... );
  commit;
end;
```

✓

Thanks to Steven Feuerstein for providing this example

Potential Performance Issues

#2 – execute a single DML inside a loop

all or nothing approach

```
begin

  insert into LICENSEE_ADDRESS (
       id,
       <whatever this one is>,
       addr1,
       addr2,
       ... )
  select
       id,
       '3',
       addr_1,
       addr_2,
       ...
  from (...)
  where (...);

  commit;

end;
```

insert what‘s possible – handle errors afterwards

```
declare
  cursor c_addresses is
    select id,addr_1,addr_2,city,state_code_id,
               zip,zip_plus4,country_id,updated_date,updated_by
     from (...)
    where (...);

  type addresses_t is table of c_addresses%rowtype;
  l_addr addresses_t;
begin
  open c_addresses;
  fetch c_addresses bulk collect into l_addr;
  close c_addresses;

  forall indx in 1 .. l_addr.count
    insert into LICENSEE_ADDRESS ( ... )
     values ( ... );
  commit;









end;
```

Thanks to Steven Feuerstein for providing this example

TRIOLOGY

Potential Performance Issues

#2 – execute a single DML inside a loop

all or nothing approach

```
begin

  insert into LICENSEE_ADDRESS (
       id,
       <whatever this one is>,
       addr1,
       addr2,
       ... )
  select
       id,
       '3',
       addr_1,
       addr_2,
       ...
  from (...)
  where (...);

  commit;

end;
```

insert what‘s possible – handle errors afterwards

```
declare
  cursor c_addresses is
    select id,addr_1,addr_2,city,state_code_id,
               zip,zip_plus4,country_id,updated_date,updated_by
     from (...)
     where (...);

  type addresses_t is table of c_addresses%rowtype;
  l_addr addresses_t;
begin
  open c_addresses;
  fetch c_addresses bulk collect into l_addr;
  close c_addresses;

  forall indx in 1 .. l_addr.count SAVE EXCEPTIONS
    insert into LICENSEE_ADDRESS ( ... )
     values ( ... );
  commit;

exception when std_errs.failure_in_forall then
  DBMS_OUTPUT.put_line (SQLERRM);
  DBMS_OUTPUT.put_line ('Inserted ' || SQL%ROWCOUNT || ' rows.');

  for indx in 1 .. SQL%BULK_EXCEPTIONS.count
  loop
    DBMS_OUTPUT.put_line (
      'Error index ' || SQL%BULK_EXCEPTIONS (indx).ERROR_INDEX || ' is - '
      || SQLERRM ( -1 * SQL%BULK_EXCEPTIONS (indx).ERROR_CODE));
  end loop;
  commit;
end;
```

Thanks to Steven Feuerstein for providing this example

TRIOLOGY

Potential Performance Issues

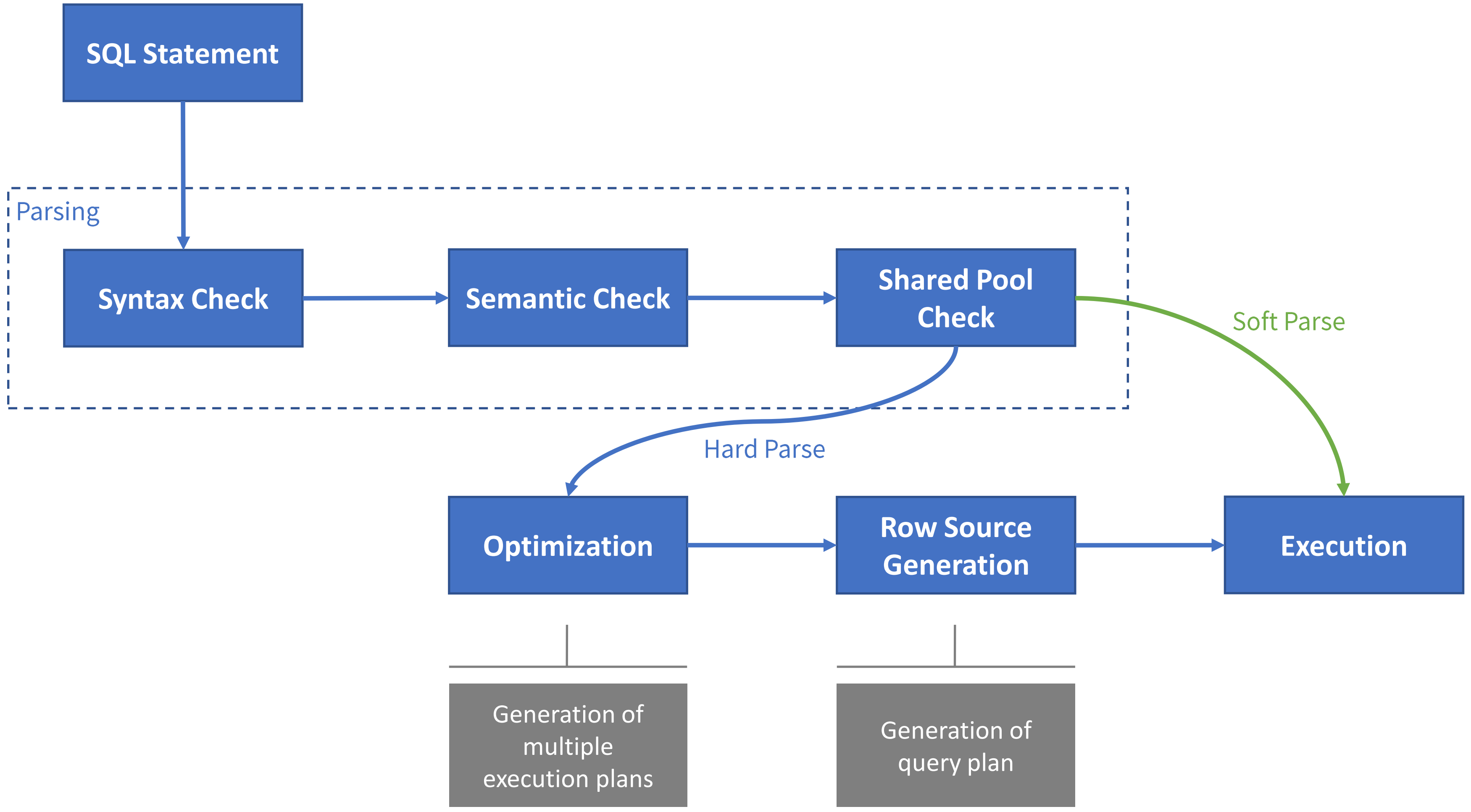
#3 – dynamic sql without bind variables

```
FUNCTION has_confirmation_doc (
    in_bt_id IN drt_businesstransaction.id%type)
RETURN NUMBER
AS
    l_count    PLS_INTEGER;

BEGIN

    execute immediate 'SELECT count(*) '            ||chr(10)||
            '  FROM drt_communication_doc cd '      ||chr(10)||
            '     ,drt_fin_document_type fdt '      ||chr(10)||
            ' WHERE cd.bt_id = ' || in_bt_id        ||chr(10)||
            '   AND fdt.id = cd.fin_type_id '       ||chr(10)||
            '   AND fdt.metavalue = ' || c_metavalue
        INTO l_count ;

    IF l_count = 0 THEN
        return 0;
    ELSE
        return 1;
    END IF;

END has_confirmation_doc;
```

TRIOLOGY

Potential Performance Issues

#3 – dynamic sql without bind variables



TRIOLOGY

Potential Performance Issues

#3 – dynamic sql without bind variables

```
function get_address(in_id in crm_addresses.id%type)
  return crm_addresses%rowtype
is
  l_sql varchar2(4000 char);
  l_address crm_addresses%rowtype;
begin
  l_sql := 'select * from crm_addresses where id = ' || in_id;
  execute immediate l_sql into l_address;
  return l_address;
end get_address;
```

```
declare
  l_address crm_addresses%rowtype;
begin
  l_address := get_address(in_id => 66);
  l_address := get_address(in_id => 73);
  l_address := get_address(in_id => 109);
end;
```

'select * from crm_addresses where id = 66’

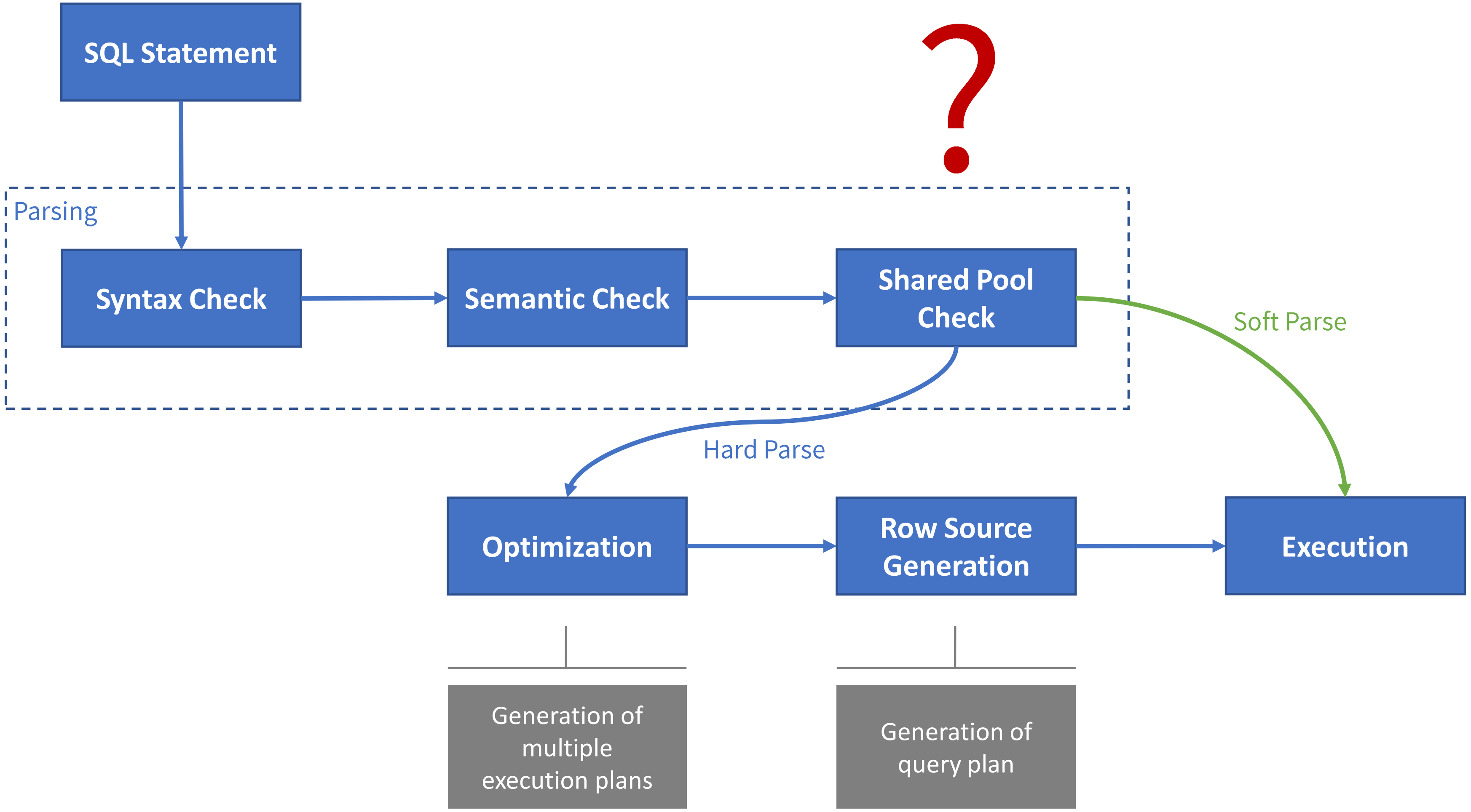
'select * from crm_addresses where id = 73’

'select * from crm_addresses where id = 109'

TRIOLOGY

Potential Performance Issues

#3 – dynamic sql without bind variables



TRIOLOGY

Potential Performance Issues

#3 – dynamic sql without bind variables

```
function get_address(in_id in crm_addresses.id%type)
  return crm_addresses%rowtype
is
  l_sql varchar2(4000 char);
  l_address crm_addresses%rowtype;
begin
  l_sql := 'select * from crm_addresses where id = :var';
  execute immediate l_sql into l_address using in_id;
  return l_address;
end get_address;
```

```
declare
  l_address crm_addresses%rowtype;
begin
  l_address := get_address(in_id => 66);
  l_address := get_address(in_id => 73);
  l_address := get_address(in_id => 109);
end;
```

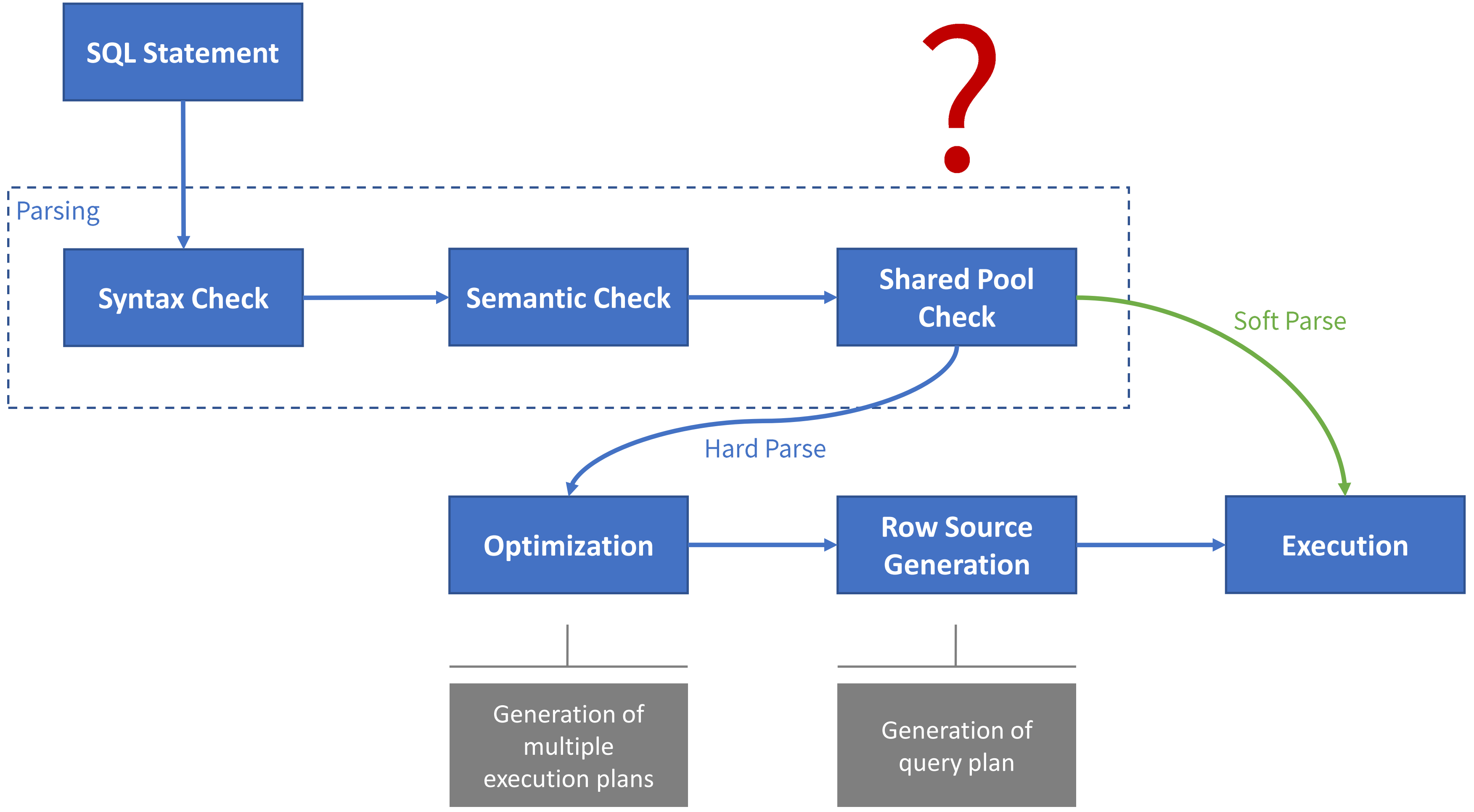
'select * from crm_addresses where id = :var’

'select * from crm_addresses where id = :var’

'select * from crm_addresses where id = :var'

Potential Performance Issues

#3 – dynamic sql without bind variables



TRIOLOGY

Potential Performance Issues

#3 – dynamic sql without bind variables

```
FUNCTION has_confirmation_doc (
    in_bt_id IN drt_businesstransaction.id%type)
RETURN NUMBER
AS
    l_count    PLS_INTEGER;

BEGIN

    execute immediate 'SELECT count(*) '            ||chr(10)||
             ' FROM drt_communication_doc cd '       ||chr(10)||
             '     ,drt_fin_document_type fdt '      ||chr(10)||
             ' WHERE cd.bt_id = ' || in_bt_id        ||chr(10)||
             '  AND fdt.id = cd.fin_type_id '        ||chr(10)||
             '  AND fdt.metavalue = ' || c_metavalue
        INTO l_count ;

    IF l_count = 0 THEN
        return 0;
    ELSE
        return 1;
    END IF;

END has_confirmation_doc;
```

good

```
FUNCTION has_confirmation_doc(
    in_bt_id IN drt_businesstransaction.id%type)
RETURN NUMBER
AS
    l_count    PLS_INTEGER;

BEGIN

    execute immediate 'SELECT count(*) '            ||chr(10)||
             ' FORM drt_communication_doc cd '       ||chr(10)||
             '     ,drt_fin_document_type fdt '      ||chr(10)||
             ' WHERE cd.bt_id = :var1 '              ||chr(10)||
             '  AND fdt.id = cd.fin_type_id '        ||chr(10)||
             '  AND fdt.metavalue = :var2'
         INTO l_count using in_bt_id, c_metavalue ;

    IF l_count = 0 THEN
        return 0;
    ELSE
        return 1;
    END IF;

END has_confirmation_doc;
```

Potential Performance Issues

#3 – dynamic sql without bind variables

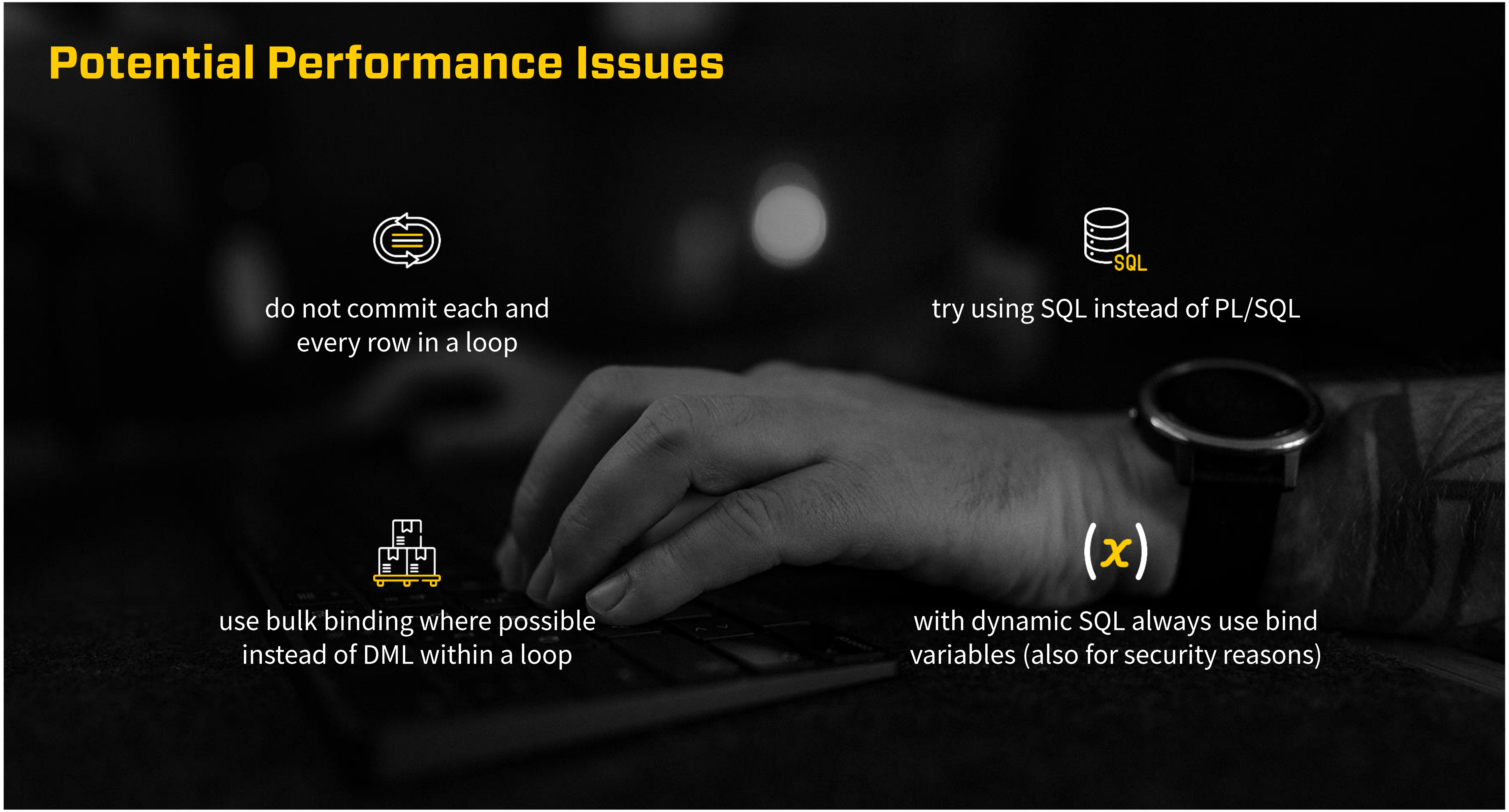
```
FUNCTION has_confirmation_doc (
    in_bt_id IN drt_businesstransaction.id%type)
RETURN NUMBER
AS
    l_count    PLS_INTEGER;

BEGIN

    execute immediate 'SELECT count(*) '          ||chr(10)||
             ' FROM drt_communication_doc cd '   ||chr(10)||
             '    ,drt_fin_document_type fdt '   ||chr(10)||
             ' WHERE cd.bt_id = ' || in_bt_id    ||chr(10)||
             '  AND fdt.id = cd.fin_type_id '    ||chr(10)||
             '  AND fdt.metavalue = ' || c_metavalue
        INTO l_count ;

    IF l_count = 0 THEN
        return 0;
    ELSE
        return 1;
    END IF;

END has_confirmation_doc;
```

better

```
FUNCTION has_confirmation_doc(
    in_bt_id IN drt_businesstransaction.id%type)
RETURN NUMBER
AS
    l_count    PLS_INTEGER;

BEGIN

    SELECT count(*)
    INTO l_count
    FROM drt_communication_doc cd
        ,drt_fin_document_type fdt
    WHERE cd.bt_id = in_bt_id
    AND fdt.id = cd.fin_type_id
    AND fdt.metavalue = c_metavalue;

    IF l_count = 0 THEN
        return 0;
    ELSE
        return 1;
    END IF;

END has_confirmation_doc;
```

TRIOLOGY

Potential Performance Issues



do not commit each and every row in a loop

try using SQL instead of PL/SQL

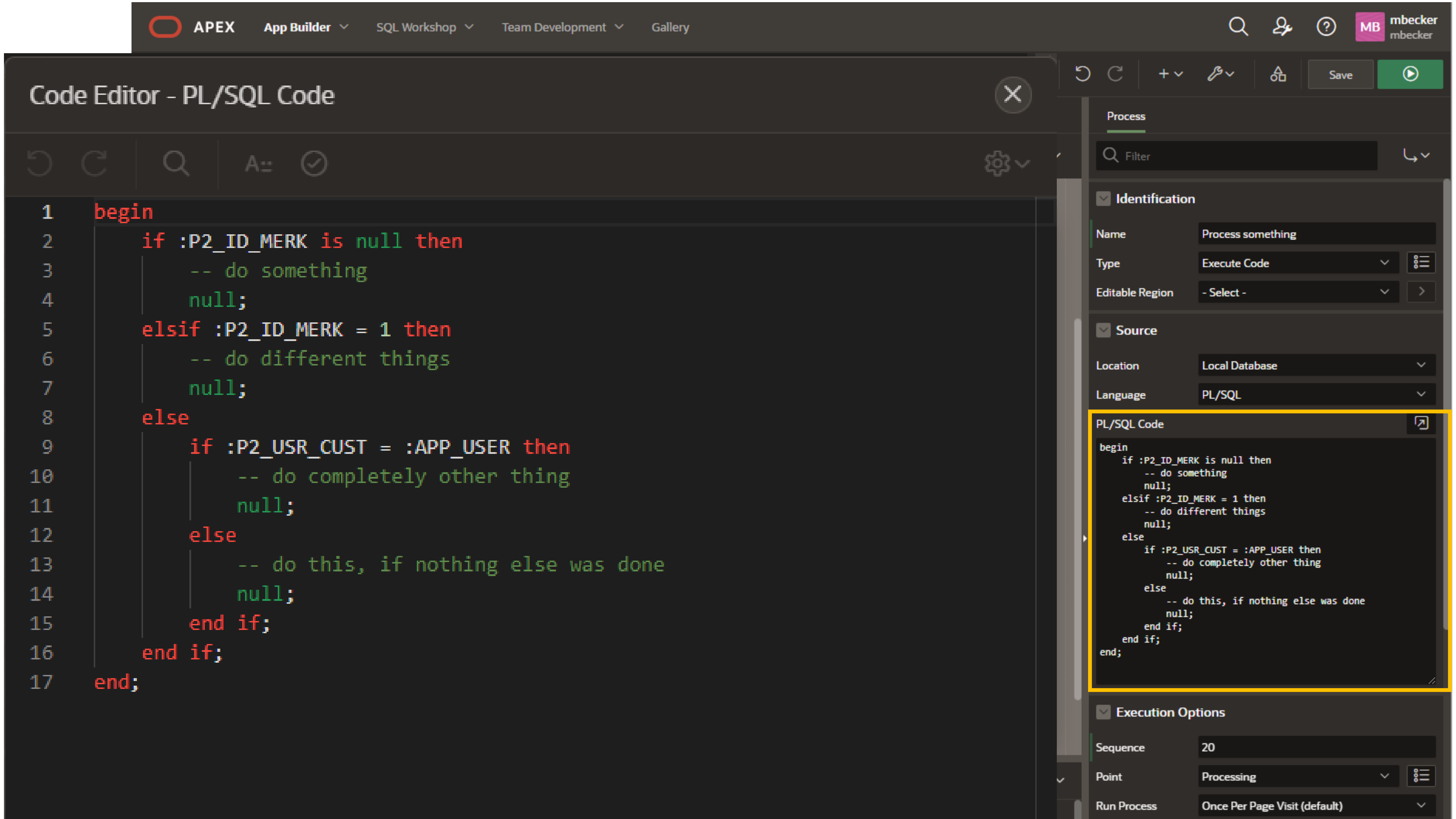
use bulk binding where possible instead of DML within a loop

with dynamic SQL always use bind variables (also for security reasons)

PL/SQL and APEX

PL/SQL and APEX

#1 – pl/sql logic in pages, processes, …



```
begin
    if :P2_ID_MERK is null then
        -- do something
        null;
    elsif :P2_ID_MERK = 1 then
        -- do different things
        null;
    else
        if :P2_USR_CUST = :APP_USER then
            -- do completely other thing
            null;
        else
            -- do this, if nothing else was done
            null;
        end if;
    end if;
end;
```

Identification

Name	Process something
Type	Execute Code
Editable Region	- Select -

Source

Location	Local Database
Language	PL/SQL

PL/SQL Code

Execution Options

Sequence	20
Point	Processing
Run Process	Once Per Page Visit (default)

TRIOLOGY

PL/SQL and APEX

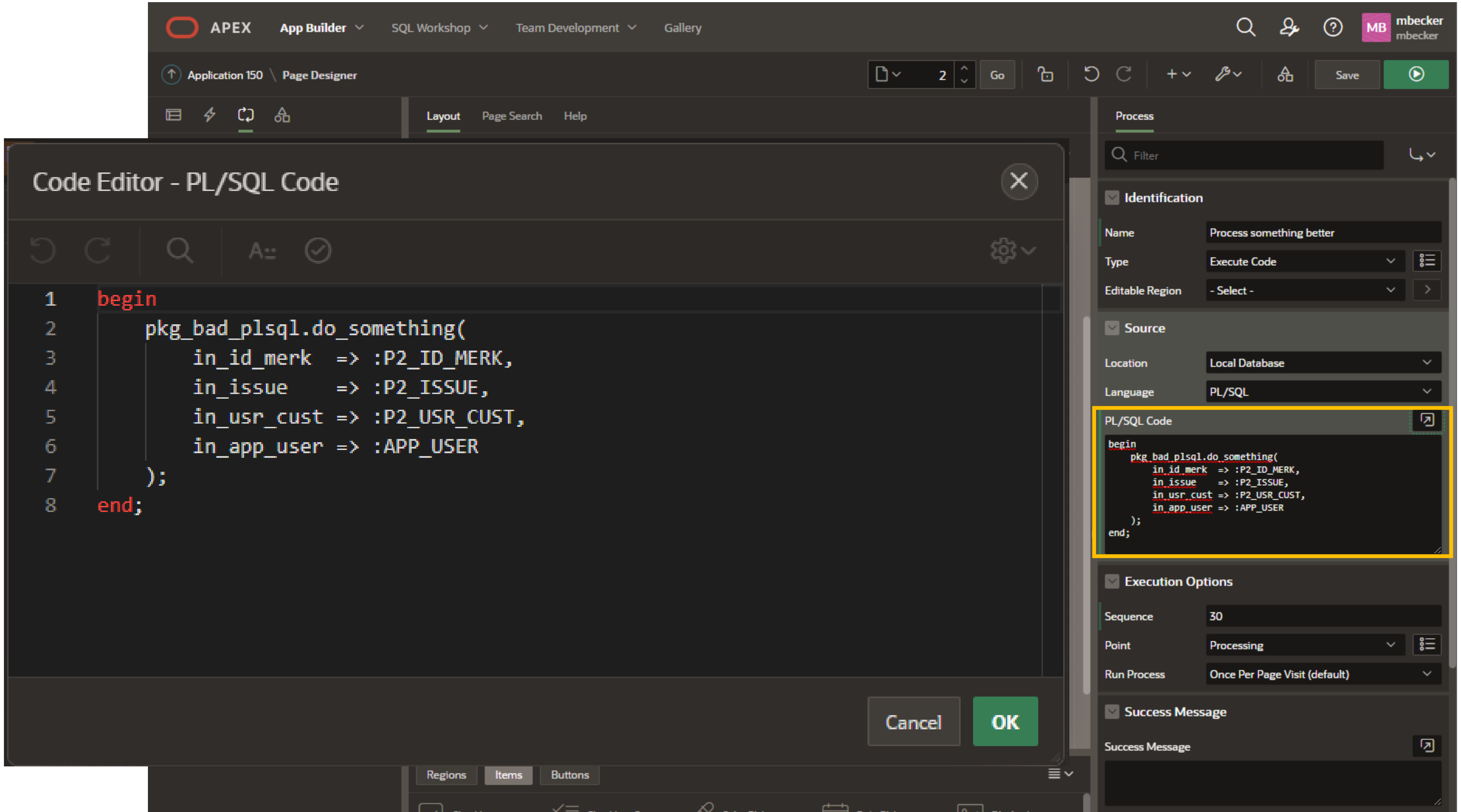
#1 – pl/sql logic in pages, processes, …

```
create or replace package body pkg_good_plsql
is
  procedure do_something(
     in_id_merk in number
    ,in_issue in varchar2
    ,in_usr_cust in varchar2
    ,in_app_user in varchar2)
  is
  begin
    if in_id_merk is null then
      -- do something
      null;
    elsif in_id_merk = 1 then
      -- do different things
      null;
    else
      if in_usr_cust = in_app_user then
        -- do completely other thing
        null;
      else
        -- do this, if nothing else was done
        null;
      end if;
    end if;
  end do_something;

end pkg_good_plsql;
```

TRIOLOGY

PL/SQL and APEX

#1 – pl/sql logic in pages, processes, …



```
begin
    pkg_bad_plsql.do_something(
        in_id_merk   => :P2_ID_MERK,
        in_issue     => :P2_ISSUE,
        in_usr_cust => :P2_USR_CUST,
        in_app_user => :APP_USER
    );
end;
```

Identification

Name	Process something better
Type	Execute Code
Editable Region	- Select -

Source

Location	Local Database
Language	PL/SQL

Execution Options

Sequence	30
Point	Processing
Run Process	Once Per Page Visit (default)

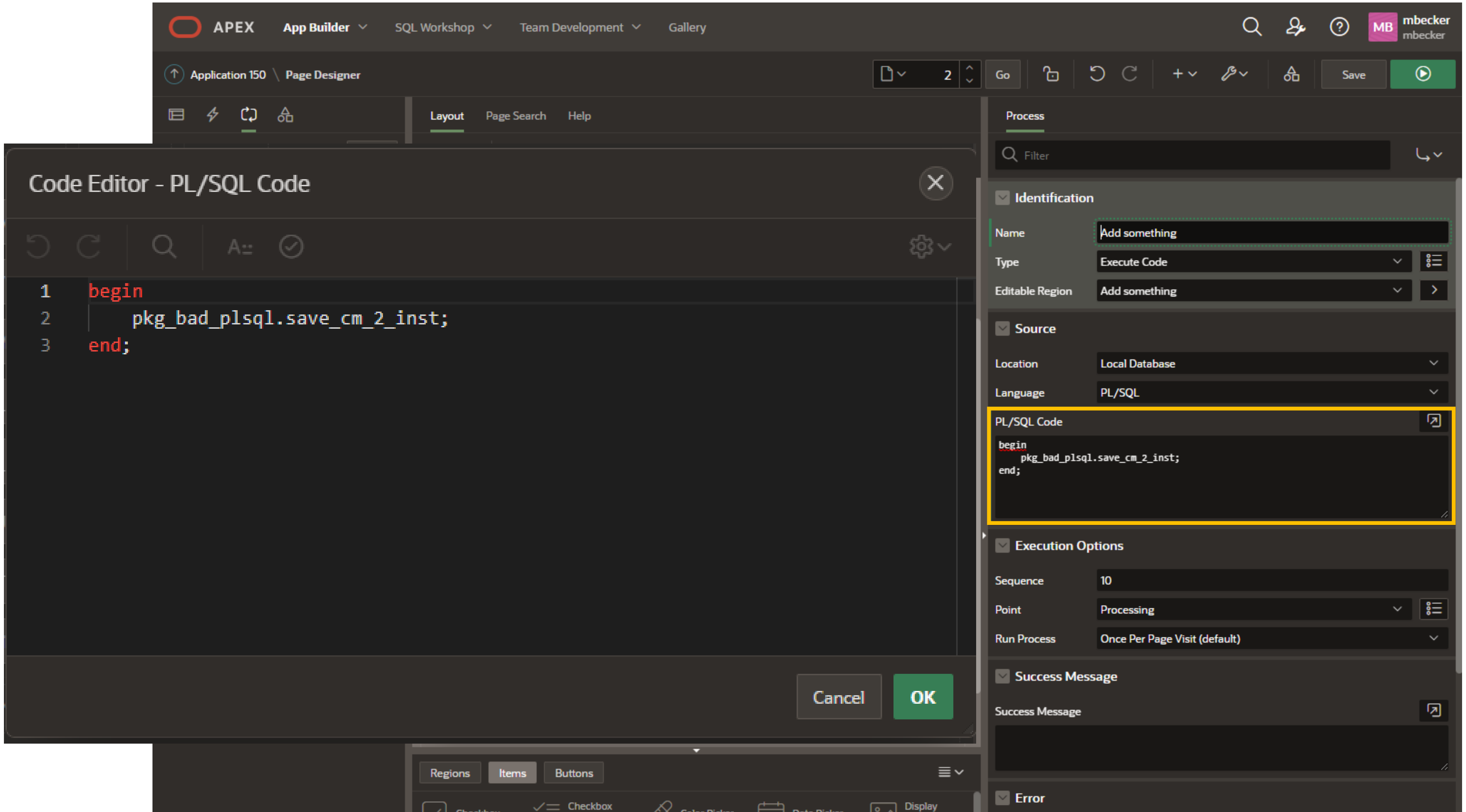
Success Message

Success Message

TRIOLOGY

PL/SQL and APEX

#2 – using v() to get page item values in a pl/sql program



```
begin
    pkg_bad_plsql.save_cm_2_inst;
end;
```

Identification

Name	Add something
Type	Execute Code
Editable Region	Add something

Source

Location	Local Database
Language	PL/SQL

PL/SQL Code

```
begin
    pkg_bad_plsql.save_cm_2_inst;
end;
```

Execution Options

Sequence	10
Point	Processing
Run Process	Once Per Page Visit (default)

Success Message

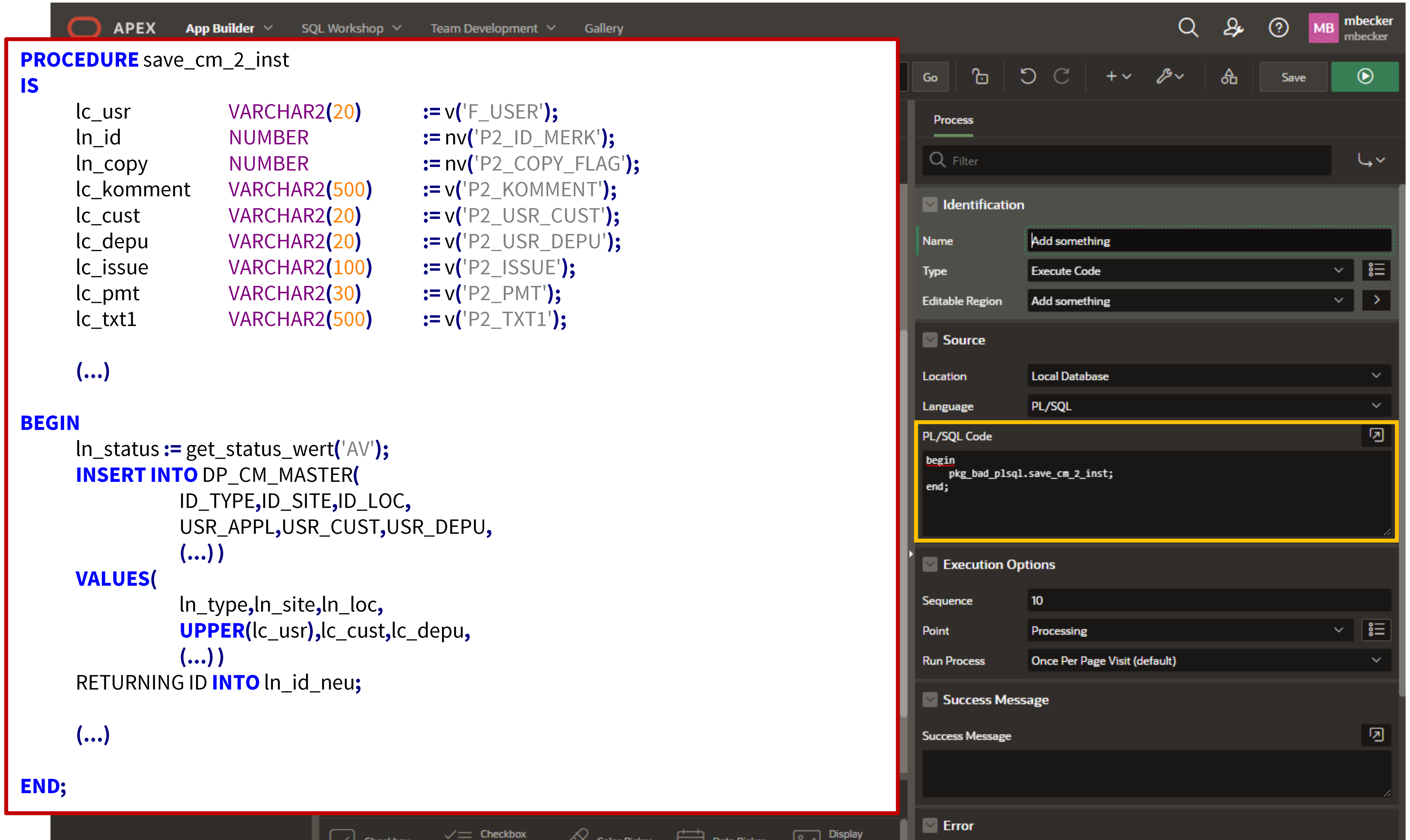
Success Message

Error

TRIOLOGY

PL/SQL and APEX

#2 – using v() to get page item values in a pl/sql program



```
PROCEDURE save_cm_2_inst
IS
        lc_usr          VARCHAR2(20)        := v('F_USER');
        ln_id           NUMBER              := nv('P2_ID_MERK');
        ln_copy         NUMBER              := nv('P2_COPY_FLAG');
        lc_komment      VARCHAR2(500)       := v('P2_KOMMENT');
        lc_cust         VARCHAR2(20)        := v('P2_USR_CUST');
        lc_depu         VARCHAR2(20)        := v('P2_USR_DEPU');
        lc_issue        VARCHAR2(100)       := v('P2_ISSUE');
        lc_pmt          VARCHAR2(30)        := v('P2_PMT');
        lc_txt1         VARCHAR2(500)       := v('P2_TXT1');

        (...)

BEGIN
        ln_status := get_status_wert('AV');
        INSERT INTO DP_CM_MASTER(
                        ID_TYPE,ID_SITE,ID_LOC,
                        USR_APPL,USR_CUST,USR_DEPU,
                        (...) )
        VALUES(
                        ln_type,ln_site,ln_loc,
                        UPPER(lc_usr),lc_cust,lc_depu,
                        (...) )
        RETURNING ID INTO ln_id_neu;

        (...)

END;
```

Identification
Name: Add something
Type: Execute Code
Editable Region: Add something

Source
Location: Local Database
Language: PL/SQL
PL/SQL Code:

```
begin
    pkg_bad_plsql.save_cm_2_inst;
end;
```

Execution Options
Sequence: 10
Point: Processing
Run Process: Once Per Page Visit (default)

Success Message
Success Message:

Error

TRIOLOGY

PL/SQL and APEX

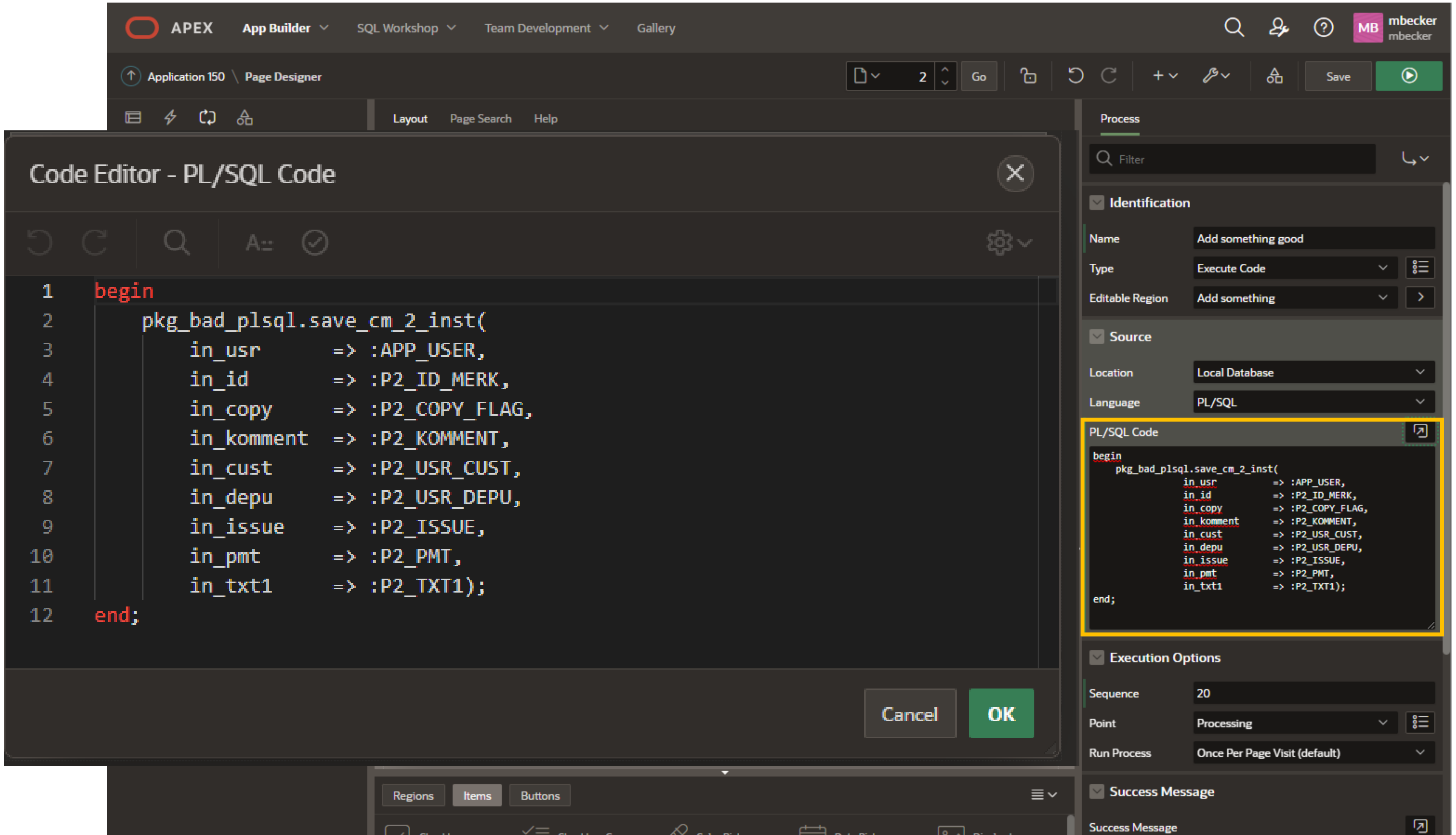
#2 – using v() to get page item values in a pl/sql program

```
PROCEDURE save_cm_2_inst(
          in_usr            IN      VARCHAR2,
          in_id             IN      NUMBER,
          in_copy           IN      NUMBER,
          in_komment        IN      VARCHAR2,
          in_cust           IN      VARCHAR2,
          in_depu           IN      VARCHAR2,
          in_issue          IN      VARCHAR2,
          in_pmt            IN      VARCHAR2,
          in_txt1           IN      VARCHAR2,
          (...)
          )
IS
          (...)
BEGIN
          INSERT INTO DP_CM_MASTER(
                            ID_TYPE,ID_SITE,ID_LOC,
                            USR_APPL,USR_CUST,USR_DEPU,
                            (...) )
          VALUES(
                            ln_type,ln_site,ln_loc,
                            UPPER(in_usr),in_cust,in_depu,
                            (...) )
          RETURNING ID INTO ln_id_neu;

          (...)

END save_cm_2_inst;
```

TRIOLOGY

PL/SQL and APEX

#2 – using v() to get page item values in a pl/sql program



```
begin
    pkg_bad_plsql.save_cm_2_inst(
        in_usr       => :APP_USER,
        in_id        => :P2_ID_MERK,
        in_copy      => :P2_COPY_FLAG,
        in_komment   => :P2_KOMMENT,
        in_cust      => :P2_USR_CUST,
        in_depu      => :P2_USR_DEPU,
        in_issue     => :P2_ISSUE,
        in_pmt       => :P2_PMT,
        in_txt1      => :P2_TXT1);
end;
```

Identification

Name	Add something good
Type	Execute Code
Editable Region	Add something

Source

Location	Local Database
Language	PL/SQL

PL/SQL Code

Execution Options

Sequence	20
Point	Processing
Run Process	Once Per Page Visit (default)

Success Message

Success Message

TRIOLOGY

PL/SQL and APEX

#3 – set items using set_session_state in a pl/sql program

```
PROCEDURE sel_cm_17_one
IS
        lc_status    VARCHAR2(50);
        lc_issue     VARCHAR2(120);
        ln_type      NUMBER;
        lc_files     VARCHAR2(60);
BEGIN
        -- do some stuff here

        -- then select some data
        SELECT (...) INTO lc_status, lc_issue (...) FROM (...) WHERE (...) ;

        -- and then ...
        ~~htmldb_util.SET_SESSION_STATE('P17_CM_STATUS', lc_status);~~
        ~~htmldb_util.SET_SESSION_STATE('P17_ISSUE_TEXT', lc_issue);~~
        ~~htmldb_util.SET_SESSION_STATE('P17_TYPE', ln_type);~~
        ~~htmldb_util.SET_SESSION_STATE('P17_FILE_BASE', lc_files);~~

END sel_cm_17_one;
```

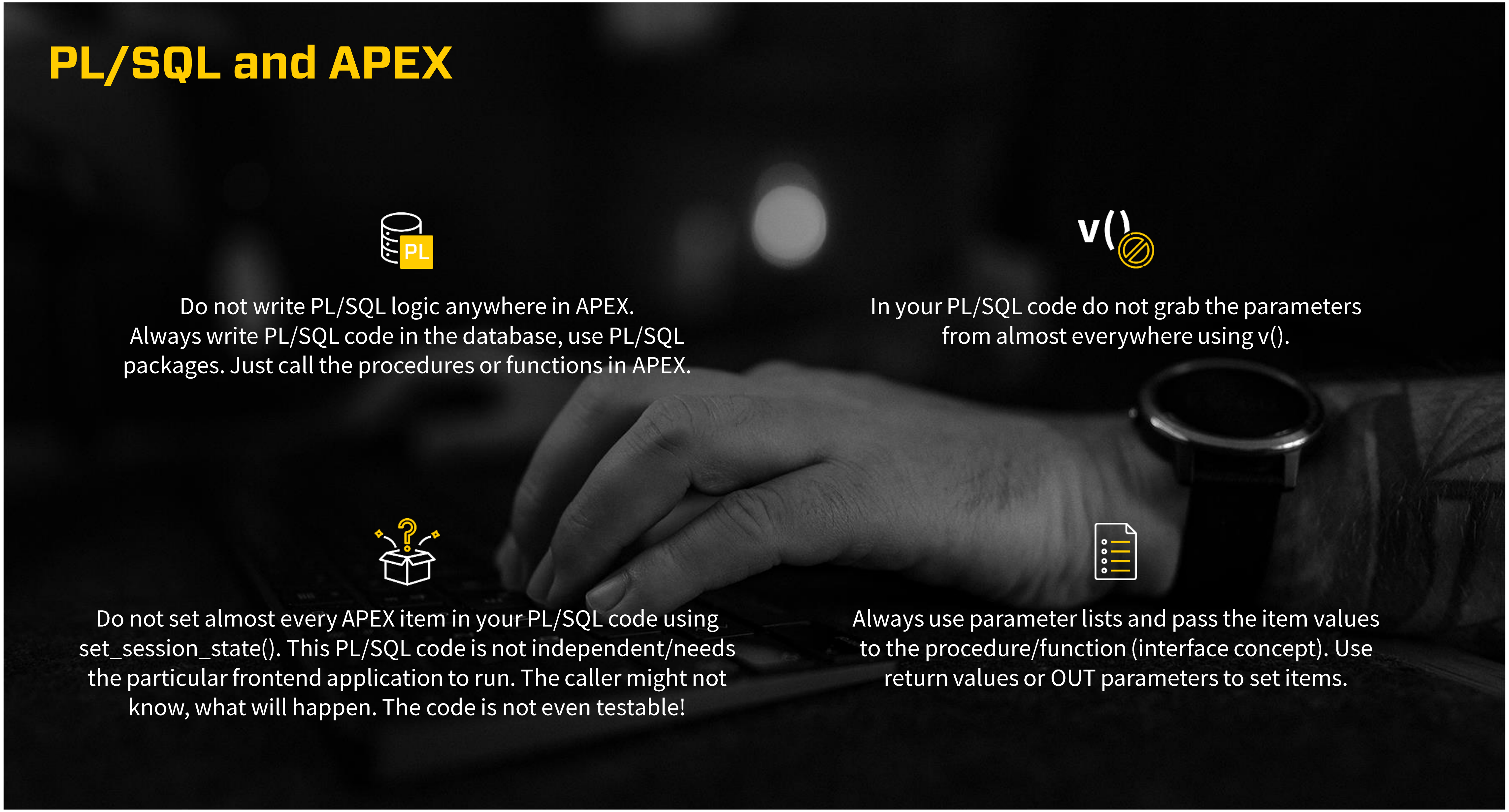
Better:

- ✓ use views to select data …
- ✓ use table functions to select data …
- ✓ use return values to get data …
- ✓ use out parameters to get data …
- ✓ use complex types as return values …

- ✓ … and set your items using these values!

TRIOLOGY

PL/SQL and APEX



Do not write PL/SQL logic anywhere in APEX.
Always write PL/SQL code in the database, use PL/SQL packages. Just call the procedures or functions in APEX.

In your PL/SQL code do not grab the parameters from almost everywhere using v().

Do not set almost every APEX item in your PL/SQL code using set_session_state(). This PL/SQL code is not independent/needs the particular frontend application to run. The caller might not know, what will happen. The code is not even testable!

Always use parameter lists and pass the item values to the procedure/function (interface concept). Use return values or OUT parameters to set items.

Maik Becker

maik.becker@triology.de

@beckerman_maik

Business Unit Manager @TRIOLOGY

> 20 years of experience in software development

> 10 years of experience with Oracle DB and APEX

Oracle ACE Associate

Alexandra Welzel

alexandra.welzel@triology.de

@alexandrawelzel

Data Consultant @TRIOLOGY

focused on Oracle DB and Oracle APEX

6 years of experience in database development

Love Oracle ā'pěks